Université de Montréal

Incorporating Complex Cells into Neural Networks for Pattern Classification

par
James Bergstra

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en Informatique

11 Mars, 2011

Université de Montréal
Faculté des arts et des sciences

Cette thèse intitulée:

Incorporating Complex Cells into Neural Networks for Pattern Classification

présentée par:

James Bergstra

a été évaluée par un jury composé des personnes suivantes:

Pascal Vincent,	président-rapporteur
Yoshua Bengio,	directeur de recherche
Paul Cisek,	membre du jury
Bruno Olshausen,	examinateur externe
Christian Léger,	représentant du doyen de la FESP

Thèse acceptée le: .....

RÉSUMÉ

Dans le domaine des neurosciences computationnelles, l'hypothèse a été émise que le système visuel, depuis la rétine et jusqu'au cortex visuel primaire au moins, ajuste continuellement un modèle probabiliste avec des variables latentes, à son flux de perceptions. Ni le modèle exact, ni la méthode exacte utilisée pour l'ajustement ne sont connus, mais les algorithmes existants qui permettent l'ajustement de tels modèles ont besoin de faire une estimation conditionnelle des variables latentes. Cela nous peut nous aider à comprendre *pourquoi* le système visuel pourrait ajuster un tel modèle ; si le modèle est approprié, ces estimé conditionnels peuvent aussi former une excellente représentation, qui permettent d'analyser le contenu sémantique des images perçues. Le travail présenté ici utilise la performance en classification d'images (discrimination entre des types d'objets communs) comme base pour comparer des modèles du système visuel, et des algorithmes pour ajuster ces modèles (vus comme des densités de probabilité) à des images. Cette thèse (a) montre que des modèles basés sur les *cellules complexes* de l'aire visuelle V1 généralisent mieux à partir d'exemples d'entraînement étiquetés que les réseaux de neurones conventionnels, dont les unités cachées sont plus semblables aux *cellules simples* de V1 ; (b) présente une nouvelle interprétation des modèles du système visuels basés sur des cellules complexes, comme distributions de probabilités, ainsi que de nouveaux algorithmes pour les ajuster à des données ; et (c) montre que ces modèles forment des représentations qui sont meilleures pour la classification d'images, après avoir été entraînés comme des modèles de probabilités.

Deux innovations techniques additionnelles, qui ont rendu ce travail possible, sont également décrites : un algorithme de recherche aléatoire pour sélectionner des hyper-paramètres, et un compilateur pour des expressions mathématiques matricielles, qui peut optimiser ces expressions pour processeur central (CPU) et graphique (GPU).

Mots clés: apprentissage machine, aire visuelle V1, selection d'hyper-paramètres, vision numerique, vision biologique.

ABSTRACT

Computational neuroscientists have hypothesized that the visual system from the retina to at least primary visual cortex is continuously fitting a latent variable probability model to its stream of perceptions. It is not known exactly which probability model, nor exactly how the fitting takes place, but known algorithms for fitting such models require conditional estimates of the latent variables. This gives us a strong hint as to *why* the visual system might be fitting such a model; in the right kind of model those conditional estimates can also serve as excellent features for analyzing the semantic content of images perceived. The work presented here uses image classification performance (accurate discrimination between common classes of objects) as a basis for comparing visual system models, and algorithms for fitting those models as probability densities to images. This dissertation (a) finds that models based on visual area V1's *complex cells* generalize better from labeled training examples than conventional neural networks whose hidden units are more like V1's *simple cells*, (b) presents novel interpretations for complex-cell-based visual system models as probability distributions and novel algorithms for fitting them to data, and (c) demonstrates that these models form better features for image classification after they are first trained as probability models. Visual system models based on complex cells achieve some of the best results to date on the CIFAR-10 image classification benchmark, and samples from their probability distributions indicate that they have learnt to capture important aspects of natural images.

Two auxiliary technical innovations that made this work possible are also described: a random search algorithm for selecting hyper-parameters, and an optimizing compiler for matrix-valued mathematical expressions which can target both CPU and GPU devices.

Keywords: machine learning, visual area V1, hyper-parameter selection, computer vision, biological vision.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CD	Contrastive Divergence, also known as CD-1
CD-K	Contrastive Divergence with K Gibbs steps
DAE	Denoising Auto-Encoder
GMM	Gaussian Mixture Model
GRBM	Gaussian Restricted Boltzmann Machine
ICA	Independent Components Analysis
IT	Inferotemporal lobe
LGN	Lateral Geniculate Nucleus
mcRBM	Mean and Covariance RBM
MT	Medial Temporal lobe
NLL	Negative Log-Likelihood
PCA	Principal Components Analysis
PCD	Persistent Contastive Divergence
RBM	Restricted Boltzmann Machine
RGB	Red Green Blue
RI	Real Intelligence
SIFT	Scale Invariant Feature Transform
SFA	Slow Feature Analysis
SPD	Sparse Predictive Decomposition
SGD	Stochastic Gradient Descent
SML	Stochastic Maximum Likelihood
ssRBM	Spike and Slab RBM
SVM	Support Vector Machine
V1,V2,V4	Visual area 1, 2, 4

NOTATION

A	learning algorithm
$\mathbb{B}$	binary values, the set $\{0,1\}$
$\mathbb{C}$	complex numbers
E	(italic) energy, energy function
$\mathbb{E}[\cdot]$, $\mathbb{E}_Q[\cdot]$	(no italic) expectation under true distribution, distribution Q
$\mathscr{G}_x$	true, naturally occurring distribution of observable variable x
$\mathbb{I}$	indicator function, e.g. $\mathbb{I}_{(x>0)} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$
$\mathbb{N}$	natural numbers
$\mathscr{N}(\mu, \sigma^2)$	a Normal distribution of mean μ and variance σ^2
$O(f(n))$	big-O notation for asymptotic space or time requirement
$P(\cdot), P_Q(\cdot)$	probability under true distribution, distribution Q
$\mathbb{R}$	real numbers
u, v, w, x, y, z	lower case latin letters - vectors
U, V, W, X, Y, Z	upper case latin letters - matrices
v_i, V_i	i^{th} element of vector v, column of matrix V
$\alpha, \beta, \gamma, \delta, \varepsilon$	real-valued scalars
$\mathscr{L}^{(\mathrm{A})}(x; \theta)$	loss function A evaluated at x with model parameters θ
$\mathscr{X}, \mathscr{Y}$	caligraphic font indicates a dataset, a sample from some $\mathscr{G}_x, \mathscr{G}_y$
$\lvert\mathscr{X}\rvert$	cardinality of dataset
$\mathrm{mean}_{a \in \mathscr{X}}(f(a))$	mean of f over $\mathscr{X}$, i.e. $\frac{1}{\lvert\mathscr{X}\rvert} \sum_{a \in \mathscr{X}} f(a)$
$\mathrm{Var}_{a \in \mathscr{X}}(f(a))$	variance in f over $\mathscr{X}$, otherwise similar to mean.
$\lvert\lvert x \rvert\rvert_1$	The ℓ_1-norm of $x \in \mathbb{R}^N$, $\sum_{i=1}^N \lvert x_i \rvert$
$\lvert\lvert x \rvert\rvert_2$	The ℓ_2-norm of $x \in \mathbb{R}^N$, $\sqrt{\sum_{i=1}^N \lvert x_i \rvert^2}$
$U * W$	The 2D convolution of image U with filter W

ACKNOWLEDGMENTS

The events of life are so intricately interwoven, and my view of that web so narrow, that I will never know how many people helped me to complete this dissertation, nor how much they put up with as I slogged my way through it. I know there were at least a few who put up with quite a bit. I would like to specifically thank Yoshua Bengio, Aaron Courville, Pascal Vincent, Olivier Breuleux, Frédéric Bastien, Dumitru Erhan, Guillaume Desjardins, Pierre-Antoine Manzagol, François Rivest, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Joseph Turian, Antoine Bordes, Michael Mandel, Douglas Eck, Jérôme Louradour, Paul Cisek, Andrea Green, David Warde-Farley, Ian Goodfellow, Nicolas LeRoux, and Hugo Larochelle for offering their insight, many interesting discussions, and of course the time and effort they invested in Theano. I would like to thank my family - Susan, Stuart, Evan, and Findlay - who were all so supportive. And I would like to especially and profoundly thank Olga, who offered her sanity and soul when I was in need, and shared the joyful moments as they came too.

CHAPTER 1

INTRODUCTION

Barlow (1961) made what has become a very influential claim regarding how the brain works. After studying the retinal cells of frogs, Barlow suggested that they seemed to be operating as if to minimize redundancy. Since then, a lot of evidence has accumulated supporting the idea that the visual system is organizing itself around statistics of what the eyes see. For example, gazing on horizontal and then vertical stripes for a few minutes will provoke your visual system to suffer from an optical illusion for weeks afterwards (McCollough, 1965). A more dramatic example comes from a recent experiment, in which the brains of baby ferrets were rewired at birth so that the eyes directed their signals to what is usually auditory cortex (Von Melchner et al., 2000). Astonishingly, the ferrets developed with a sense of sight. The implication is that the cortex of the brain is in fact a very flexible mass of learning material, just waiting to pick up on certain kinds of statistical patterns wherever they arise. If we could only discover how the brain models images, then we might well discover how it processes and represents sound and other sensory signals too. Other experiments such as Cox et al. (2005); Farley et al. (2007); Stevenson et al. (2010) suggest that our brain is constantly adapting to statistical patterns throughout our lifetime.

It is not known exactly what sort of probability model the visual system builds of the images that it sees. Many hypotheses have been advanced (such as Bell and Sejnowski, 1997; Berkes and Wiskott, 2005; Field, 1994; Hinton, 2007; Hyvärinen et al., 2001; Karklin and Lewicki, 2008; Kohonen, 1996; Lyu and Simoncelli, 2007; Olshausen and Field, 1996), and they are all consistent with what is known about the visual system. For example, neurophysiological investigation has revealed that many neurons in the visual system act like simple edge detectors (Hubel and Wiesel, 1962), and every one of these models can account for that. The background information presented in Chapter 2 will describe some of these hypotheses, and my

own work in Chapters 5 and 6 advances yet another one. More accurate characterizations of neural behaviour might narrow down the list of candidate hypotheses (and such research is underway) but the computational differences between these candidates can be quite subtle and it may be a long time before we gain sufficient insight into the long-term computational properties of neural populations to rule out any of these candidates.

There is another way to evaluate hypotheses of self-organization in the visual system, beyond direct comparison to neural behaviour. By changing the level of our analysis (as described by Anderson, 1990), we can note that the *purpose* of visual system is first and foremost to inform decisions, not to putter away building a probability model of images. If we view the building of probability models as a means to an end, then it is sensible to ask which hypothesis for learning in the visual system *works best*, as a provider of image features for behavioural decisions.

Looking at the visual system in terms of being optimal for some behavioural purpose puts my research in the domain of artificial intelligence and more specifically machine learning. Alan Turing famously defined artificial intelligence (AI) as a decision making process that could animate an impostor posing as an intelligent agent (Turing, 1950). This definition is itself rooted in the more fundamental idea that scientific knowledge is based on the refutation of hypotheses rather than deduction of corollaries (Popper, 1935). For our purposes, it means that AI is a computer simulation indistinguishable from real intelligence (RI). This definition can be formalized with a mathematical function that compares AI behaviour to RI behaviour and returns a non-negative number that is 0 if and only if they are indistinguishable. In machine learning we call this a *risk function*. Machine learning is the study of how to minimize the *expected risk*, which is a weighted average risk over some set of situations where we expect our AI simulation to behave intelligently. To put this definition in terms of information processing in the visual system, a "situation" is the presentation of an image, and an example "risk function" would be 0 if the AI and RI label the image identically and otherwise 1. In machine learning we often restrict the domain of situations faced by our AI agents. The greater the

variety of situations that we consider, and the lower the expected risk, the closer our AI is to real intelligence. The principles and methods of machine learning will be reviewed in Chapter 2.

The idea that a model of the visual system should provide a useful representation of images is central to the work presented in this dissertation. Figure 1.1 illustrates in broad strokes the scope of my visual system models within the whole sequence of cortical information processing: the path from an image stimulus to a decision regarding the name of the principle visible object. This pathway exists quite literally in the brain. Neural activity in a brain region called inferotemporal (IT) cortex is activated about 100 milliseconds after the presentation of an image (after about 5-10 neurons have passed it along), and the pattern of activation there is highly correlated with the names of visible objects (Hung et al., 2005; Kreiman et al., 2006; Sato et al., 1980). The brains of many species of mammals appear to have this capacity. My dissertation presents models of this pathway through the visual system, and simulates how it adapts to statistics of the images that it perceives.

The connection between statistical modelling of sensory input and our ability to make intelligent decisions based on that input is not new, rather it is the subject of a branch of machine learning called *deep learning* (Bengio, 2009; Hinton, 2007; Hinton et al., 2006). Using simplified models of the visual system, simulations have shown that the process of adapting to low-level image statistics (known as *unsupervised learning*) in fact increases the accuracy of the higher-level process of learning to discriminate between shapes, and improves its ability to identify important cues in the images (Erhan et al., 2009; Hinton et al., 2006). My doctoral work strengthens this connection by showing (a) that low-level adaptation helps high-level performance more so in less-simplified models of the visual system, and (b) that more faithful models of the visual system also realize higher levels of performance. The visual system models I investigate are still quite simplified, but they include biologically-inspired computational elements based on *complex cells*, described in Section 2.3.4 and Chapter 3. This dissertation argues firstly that com-

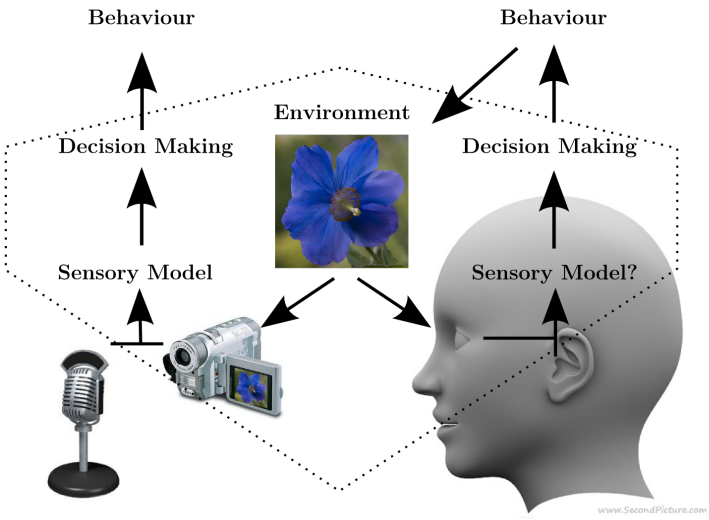

Figure 1.1: A high-level diagram of how information is processed by the nervous system. This thesis focuses on the region enclosed by the dashed line, from the perception of images to simple decisions about those images. Sensory models are probability densities over stimuli, whose latent variables are the features used to make decisions. There is substantial evidence that the brain also fits a probability density to its stream of sensory stimuli.

plex cells are useful for pattern classification even without a low-level adaptation mechanism (Chapter 3), and then shows that they can be greatly improved by such mechanisms (Chapters 4, 5, and 6). Others such as Kavukcuoglu et al. (2009) and Lee et al. (2009) have looked at this question concurrently using different techniques, and have come to similar conclusions.

Two auxiliary innovations were crucial for carrying out this research: a technique for model search and a math expression compiler. The visual system models I use in simulation experiments have many configuration options, and these can be tuned to improve performance. My model is not unique in this regard; most learning algorithms have these so-called *hyper-parameters* (defined and discussed in Section 2.5.6) and the standard technique for tuning performance is grid search. I found that purely random search gives much better results. I believe that there is still a great opportunity for further improvement in model search, but currently random search is arguably the state of the art in hyper-parameter optimization. It is also a much more convenient baseline for comparison than grid search.

The second innovation is *Theano*, a math expression compiler. To appreciate the value of this compiler, one must first appreciate the importance of investigating machine learning algorithms at scale - in terms of model size, data dimensionality, and number of training examples. Some algorithms perform well in few dimensions with few examples but scale poorly; some others that are mediocre on small datasets shine in settings with lots of data. The only real examples we have of effective vision systems are brains, and the representations in the visual cortex are much larger than the models that can be studied with commodity computers. Therefore, in our research we push at the limits of what is computationally possible to get a better picture of how learning and inference might work in brain-sized models, on retina-size images.

Theano is an optimizing compiler for math expressions that separates the business of running algorithms at top speed from the more scientific work of designing those algorithms. Programming an algorithm to run at top speed on a particular computer can be tedious. Programming an algorithm to run at top speed on

a variety of computers is harder still, because of differences in the hardware and libraries available on different computers. Theano has been essential in supporting an easy transition (for many others as well as myself) from primarily CPU-based simulations to primarily GPU-based simulations. [1] Also, its support for symbolic differentiation has made it a great tool for the rapid and correct implementation of complicated neural network models.

During my doctoral studies, I was involved in a number of other research projects that are not chapters in this dissertation:

1. *An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation* (Larochelle et al., 2007).
2. *Quadratic Features and Deep Architectures for Chunking* (Turian et al., 2009).
3. *Scalable Genre and Tag Prediction with Spectral Covariance* (Bergstra et al., 2010c).
4. *Quadratic Polynomials Learn Better Image Features* (Bergstra et al., 2009).
5. *Factored Sparse Coding* (Bergstra et al., 2010b).
6. *Deep Learning in Python with Theano* (Bergstra et al., 2011b),

The first two were primarily others' projects, the third follows up on my Masters work on music classification, and the fourth has significant overlap with the ideas (though not the experiments) in Chapter 3. The fifth is an ongoing project described briefly as future work (Chapter 9). The sixth is a project in preparation for the Journal of Machine Learning Research. Together with other members of the lab, we have assembled tutorials for several machine learning algorithms based on Theano. These *Deep Learning Tutorials* have already proved valuable in Yoshua Bengio's machine learning course, and we would like to advertise them more widely in the machine learning community.

1. Graphics Processing Units (GPUs) are massively parallel computing devices developed for rendering the 3D scenes of games, but recently fitted by companies such as NVidia and AMD to support scientific computations. For the sorts of computations treated in this thesis, GPUs are often faster by ten to twenty-fold than current-generation CPUs.

1.1 Annotated Overview of Chapters

Chapter 2 provides a survey of techniques for building image representations, called image *features*. Three approaches are described: reverse-engineering (neurophysiology), engineering (computer vision), and machine learning. Later chapters are research publications that assume a certain level of background knowledge. This chapter provides that background information regarding the visual system, image processing, and machine learning algorithms.

Chapter 3 is an article published in Neural Computation titled *Suitability of V1 Energy Models for Object Classification* (Bergstra et al., 2011a). This work uses gradient descent and random search techniques to evaluate many model variations that have been associated with complex cells in the neurophysiological literature in terms of whether they help to distinguish between common kinds of objects. It establishes that even without an adaptation mechanism beyond gradient optimization of decision making performance (purely supervised learning), complex cell models make better features than simple cell models. I put this article before the others because I started work on this project first. Although it was not published until recently, the main result of this article was established already by the winter of 2008. Earlier versions of this article were based on a hand-designed set of six increasingly sophisticated models that spanned from conventional sigmoidal neural networks (simple cell-like models) to a complex cell model suggested in Rust et al. (2005). Two papers that came out in 2008 forced me to completely redo this study: Kouh and Poggio (2008) and Cox et al. (2008, later, Pinto et al. (2009)). Consequently, I think the empirical results are much stronger, and my experience with high-throughput search inspired the work on random search presented in Chapter 7.

Chapter 4 is a paper titled *Slow, Decorrelated Features for Pretraining Complex Cell Networks* (Bergstra and Bengio, 2009). It shows that unsupervised learning of the complex cell model presented in Rust et al. (2005) improves its performance in supervised learning. The algorithm used here for unsupervised learning is a com-

bination of decorrelation (a type of independent components analysis discussed in Section 2.5.16) and temporal correlation (similar in spirit to slow feature analysis, Section 2.5.15). The combination was previously suggested by Körding et al. (2004), but this paper contributes a strategy for implementing that algorithm more efficiently.

Whereas the unsupervised learning algorithm based on decorrelation and slowness learns to model images implicitly, Chapters 5 and 6 are conference papers about a complex-cell-inspired model family that learns an explicit probability distribution over the space of images. The first paper, *The Spike and Slab Restricted Boltzmann Machine* (Courville et al., 2011a) develops the basic model and shows that it works on very small image patches. The second paper, *Unsupervised Models of Images by Spike-and-Slab RBMs* (Courville et al., 2011b), scales the basic model up to somewhat larger images and describes changes to the model that improve both modelling and classification performance.

Chapter 7 is an article submitted for publication in the Journal of Machine Learning Research titled *Random Search for Hyper-parameter Optimization* (Bergstra and Bengio, 2011). It argues that random search is better than grid search for searching a model family. It follows up on the random experiments done for Chapter 3, and argues that when a simulation is not equally sensitive to all hyper-parameters, random search can be significantly more efficient than grid search. The bulk of experimental support comes from a repetition of some experiments done previously in our group, presented originally in Larochelle et al. (2007).

Chapter 8 is an article about Theano. Theano is an optimizing compiler developed to facilitate research in machine learning. *Theano: a CPU and GPU Math Expression Compiler* was presented at SciPy 2010 (Bergstra et al., 2010a), and describes the usage and scope of the software package. It goes into a little detail regarding how the software actually works, but the interested reader is referred to the online documentation and mailing lists for more up-to-date information.

Chapter 9 summarizes my findings regarding complex cells in neural networks, and outlines work in progress for scaling these approaches to larger images and image sequences.

CHAPTER 2

BACKGROUND

This chapter introduces concepts and terminology related to machine learning, computer vision, and computational neuroscience that are used in later chapters. It begins with a brief discussion of linear decision making and the role of features before going into more detailed perspectives from computer vision, neurophysiology, and machine learning regarding what those features should be. Remember that there are lists of abbreviations and notational conventions on pages xxi and xxiii.

2.1 Linear Decision Making

Linear decision making has been at the core of artificial intelligence since the beginning of the field (Fisher, 1936). This dissertation is concerned with a particular form of decision making, called classification. Classification is the problem of choosing among of several mutually exclusive labels for an item based on evidence. In this thesis the item in question is usually an image, but classification algorithms can be applied to other things too.

To classify an item o, we must characterize it by a vector $x \in \mathbb{R}^N$, whose elements x_i are called *features* of the item. A linear classifier for a problem with L labels is technically an affine function

$$Wx + b \tag{2.1}$$

from $\mathbb{R}^N \to \mathbb{R}^L$, typically parametrized by a matrix $W \in \mathbb{R}^{L \times N}$ and a *bias* $b \in \mathbb{R}^L$. What makes this generic affine function a classifier is the way it is used – we extract a class label l^* by applying an argmax operator to the L-element output vector.

$$l^* = \arg\max_{l \leq L} (Wx + b)_l \tag{2.2}$$

When there are only two possible labels we call the classifier a *binary classifier*, oth-

erwise it is called a *multiclass classifier*. Binary classifiers can be re-parametrized to need only a single vector of weights, and a single bias

$$l^* = \begin{cases} 1 & \text{if } (W_0 - W_1) \cdot x + (b_0 - b_1) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

This form was introduced as the Perceptron, and is still used in modern applications (Minksy and Papert, 1969; Rosenblatt, 1962). We will come back to the topic of linear decision rules in Section 2.5.3 when we look at algorithms for learning optimal W and b from data.

2.2 Feature Extraction

The question of what features should characterize an item for classification plays a central role throughout this thesis. There are many approaches and possibilities.

2.2.1 Direct vs. Kernel-based

There are two broad kinds of feature extraction methods, which I will call direct and kernel-based. Suppose that we have items $\{o_1, o_2, ...\} \subseteq \mathcal{O}$ that we would like to classify. Direct feature-extraction methods build a function $f : \mathcal{O} \to \mathbb{R}^N$ that maps an item to a feature vector. Kernel-based feature extraction methods instead draw on a pairwise similarity function called a *kernel* $k : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$. Once a kernel is chosen, the features of an item $o \in \mathcal{O}$ are defined by the vector of similarities to N elements of a reference set of items, which should ideally span the range of variations that might occur among items.

The work in this thesis deals exclusively with direct features, although it should be noted that kernel-based approaches to characterizing items for classification have revolutionized machine learning, especially with regards to the classification of items described by complex data structures such as molecules and web documents. Direct approaches remain more popular where good feature functions f are known,

where there has been success in *learning* feature functions from data, or where a representative reference set would be impractically large (Bengio, 2009).

2.2.2 Three Ways to Derive Image Features

Broadly speaking, three approaches have emerged for finding functions that map from images to feature vectors:

- reverse engineering,
- engineering, and
- search.

These approaches are not mutually exclusive. For example, I think it would be fair to say that the "Spike and Slab" models in Chapter 5 draws on all three. Still, the approaches imply different mindsets and algorithms for finding features. The next few paragraphs will give a quick introduction to these approaches, and then each approach will be covered in more detail. Section 2.3 will talk about the neurophysiology of the visual system (reverse engineering), Section 2.4 will describe image processing and a computer vision algorithm for feature extraction (engineering), and Section 2.5 will describe machine learning methodology with a focus on how it can be used for feature search.

2.2.2.1 Reverse Engineering

Design by reverse engineering is the approach of studying and quantifying how the brain processes visual stimuli, and of reproducing artificial systems with the same properties. Such research is typically conducted in neuroscience departments rather than computer science ones, but we can draw on the physiological findings. A lot of progress has been made in understanding how the brain processes visual stimuli, and I will give a survey of some of that progress in Section 2.3. This approach to feature design has been used recently to give very strong results on standard computer vision benchmark tasks (Pinto et al., 2008, 2009).

2.2.2.2 Engineering

Design by engineering is the approach in which we try to conceive of mathematical transformations of images that will make classification easy. In engineering features, we often think in terms of criteria such as *invariances*. We would like features that are *invariant* to changes in the image that should not change the output of a classifier, that are at the same time sensitive to image changes that might change the classifier output. Features such as "Pyramid Histogram of Oriented Gradients" (Bosch et al., 2007), "Pyramid Histogram of Words" (Lazebnik et al., 2006), "Geometric Blur" (Berg and Malik, 2001), "Scale Invariant Feature Transformation" Lowe (1999, 2004), and "Sparse Localized Features" Mutch and Lowe (2008), developed in the computer vision community are this kind of feature. We will look at the Scale Invariant Feature Transformation (SIFT) in some detail at the end of Section 2.4.

2.2.2.3 Search

Design by search is the approach of using search algorithms over function spaces to find good feature maps, based on how they perform on sets of examples. This approach to feature design is widely used, especially in the machine learning literature on neural networks (LeCun et al., 1998a; Rumelhart et al., 1986b). This is a wonderful mathematical idea, but in practice it often poses a very difficult optimization problem. Still, gradient-based optimization algorithms work to some extent, and the most effective feature-extraction approaches make what use of them they can. This approach will be described in more detail in Section 2.5.

2.2.3 Hybrid Approaches

It is natural to combine different approaches to feature learning. When parameterizing a function space that we will optimize, or choosing a density model that we will fit, it is natural to ask ourselves the same question as the feature engineer - "how might this feature function be robust or sensitive to changes in the image?"

And it is natural to look at the function found by optimization, or the distribution fit to data, and ask - "do these features behave like anything in visual cortex?" These criteria are useful in addition to raw classification ability in guiding research. Later chapters of this dissertation will often combine different approaches.

2.3 Neurophysiology of the Visual System

The visual system is the part of the nervous system that lets an organism make behavioural decisions based on input from its sense of sight, as opposed to other senses. This section describes the overall architecture of the visual system, which is consistent across a large variety of creatures. The emphasis will be on a computational interpretation of the visual system; although components of the visual system will be identified primarily in anatomical terms, it is their computational *function* (i.e. physiology) that is of primary interest. This section will summarize the computational function of the retina, the lateral geniculate nucleus (LGN), Visual Areas V1 and V2, and inferotemporal cortex (IT) in terms of the role each one plays in the recognition of objects in images. It should be noted that the visual system participates in many neural processes involved with the oculomotor system (saccades and compensation for controlled head motion), control during reaching and grasping (Pesaran et al., 2006), spatial reasoning and balance (Bear et al., 2007), and attention (Kastner and Ungerleider, 2000; Maunsell and Treue, 2006), to give a few examples. But I will only deal with the role of the visual system in recognizing objects. The analogy with algorithmic approaches to object recognition will be that the retina and LGN perform basic (though nonetheless important) signal processing of images, that V1 and V2 extract non-linear features of the processed image, and that IT acts like a classifier of the V1 and V2 features.

2.3.1 The Two Stream Hypothesis

The visual system is often modelled as two processing pipelines: the "what"-oriented *ventral stream* and the "where"-oriented *dorsal stream*. The hypothesis is

that these implement two algorithms with distinct stages of processing by which the visual signal is transformed from photon counts to semantically meaningful things such as object identities and spatial arrangement, respectively (Ungerleider and Mishkin, 1982). The distinction between the two streams has been supported by behavioural, electrophysiological and lesioning studies. It has also been supported by experiments using the Ebbinghaus illusion (Figure 2.1) that would deceive subjects when they responded based on their perception, but that would not fool them when they responded via a grasping action (Goodale and Milner, 1992). This last evidence has been disputed by Franz et al. (2000), but I personally find it compelling: if I use my fingers to estimate how big the orange circle is I think they are about right in both cases. The two-stream hypothesis supports the practice of ignoring motion as we look to the brain for features to characterize images.

2.3.2 Neurons

There are many kinds of neuron: tens of kinds in the retina alone. Most neurons have a basic structure of *dendrites* (or a dendritic arbor), a cell body (*soma*), and an *axon*. The anatomy of neurons varies widely throughout the nervous system and between species. As a general rule though, the dendrites of a neuron look something like the root system of a plant, and the axon is its stalk.

Information flows through the neuron in the direction of nutrients through the plant. Unlike plant root systems though, neurons are arranged into enormous networks in which information flows from neuron to neuron and sometimes loops back by *feedback connections*. At *synapses*, neuro-transmitters are released in bursts by an axon, so that they disrupt the electrical potential between the interior and exterior of nearby (proximal) dendrites of other neurons. These other neurons allow that electrical disturbance to migrate along the dendrites to the soma. The soma accumulates electrical changes from all dendrites, and (in most neurons) when a threshold of electric potential is reached, the soma depolarizes and triggers an electrical chain reaction called an *action potential* (or *spike*) that can propagate a long distance very quickly along the axon. The frequency at which a soma triggers

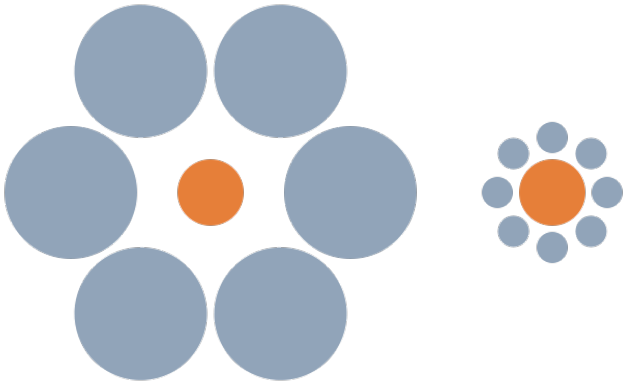

Figure 2.1: The Ebbinghaus illusion. The two orange circles are the same size, but the one on the left seems smaller. This illusion has been used to support the two-stream hypothesis of visual processing. (Image courtesy of wikipedia.)

an action potential is called the *firing rate* of the neuron. The efficiency with which a burst of neuro-transmitters disrupts the electric potential of an efferent cell is called the *synaptic strength*. It is believed that learning is the result of changes to these synaptic strengths.

While this is a widely applied theory of information processing in the visual system it is a simplification. Neurons release and take up a variety of amino acids and proteins called neuro-transmitters and neuro-modulators that change how they and their neighbours behave on various time scales. It is widely assumed among modellers that dendrites act to linearly combine the electrical disruptions mentioned above, although in many cases neurons perform non-linear computations within their dendritic arbors. There are also neurons that support dendritic spikes that propagate both to and from the soma. The kind and extent of non-linear signal processing carried out in the dendritic arbours of various neurons is the subject of active research (Carandini and Heeger, 1994; Haüsser and Mel, 2003; Heeger, 1992; Olshausen and Field, 2005; Rhodes, 2008).

2.3.3 The Retina and LGN

The retina is the part of the central nervous system in the eye, which includes photoreceptive cells. These photoreceptive cells convert light to electrical signals, much like the optic sensor in a digital camera. There are two kinds of photoreceptive cells in the retina of vertebrates: *rods* and *cones*. Rods provide luminance information and function better in faint light. Cones are tuned for colour (certain wavelengths of light) and function better in bright light. The retina fills most of the interior of the eye, and includes 75 to 150 million rods and 7 million cones. Humans typically have three kinds of cone cells, tuned to light that is roughly red, green, and blue respectively. Other species have cone cells that are tuned to different parts of the light spectrum.

Many primates including humans, some birds, and reptiles have a retina with a small central region called the *fovea* that is specialized for high-acuity vision. The central part of the fovea in humans contains only cones, and relatively few blue-sensitive cones at that, in order to maximize the visual acuity (image resolution). The rod-free area in humans is about 0.3mm in diameter and contains approximately 35,000 cones, some of which are sensitive to a *receptive field* of as little as 0.1 to 0.01 degrees (Olshausen and Field, 2005). At the fovea then, the human eye has the acuity of about a 1 mega-pixel digital camera. The non-foveal part of the retina (or sometimes the field of view) is called the *periphery*, and receptive fields there are larger (up to several degrees in diameter).

The retina communicates with the rest of the nervous system by *ganglion cells* that extend along a tract known as the *optic nerve* to the LGN. At least in higher vertebrates, this channel provides one-way communication. There are far fewer ganglion cells than there are rods, but at least in the foveal region it has been reported that cone cells map 1-to-1 or even 1-to-2 onto ganglion cells. The number of ganglion cells in mature humans is on the order of 700 thousand to 1.2 million (Curcio and Allen, 1990). Most ganglion cells are called centre-surround cells because they fire when there is more light energy in a particular visual location

relative to its surroundings, or vice-versa.

The response of these ganglion cells is modelled well by a difference-of-Gaussians band-pass filtered image (Dowling, 2007). Later we will describe this mathematical interpretation as *preprocessing* by *whitening*.

The lateral geniculate nucleus (LGN) is described as a neural relay station in the visual system (Dayan and Abbott, 2001). It routes the signals from ganglion cells to various reflex pathways, and to visual area V1.

2.3.4 Visual Area V1

The LGN projects to a relatively broad region in the posterior pole of the occipital cortex called *primary visual cortex* (V1, also *striate cortex*). Assuming that the right and left hemispheres of V1 are roughly equal in size, there are approximately 280 million neurons in V1, many more than the approximately 1-2 million afferent axons from LGN (Leuba and Kraftsik, 1994).

The neurons in V1 are divided among six functionally distinct layers. The *receptive fields* of a cell is the part of the visual field to which it is sensitive. Layer 4 receives most of the visual input from LGN. The axons from LGN project into Layer 4 in a way that reflects the topographic structure of each eye's retina, though the *retinotopic* response from the two eyes is woven together and the visual field is warped so that more V1 neurons are devoted to the fovea of the visual field (See Figure 2.2 for details). In the macaque monkey, cortical receptive fields range in size from around a tenth of a degree near the fovea to several degrees in the periphery (Dayan and Abbott, 2001). Across the striate cortex there is more functional structure than just a retinotopic organization of receptive fields. V1 neurons are organized by the retinotopic map into *columns* – groups of neurons with similar selectivity and receptive fields, which tend to be more influenced by either one eye or the other (*occular dominance*) (Carandini, 2006; Movshon et al., 1978). Figure 2.3 shows that there is a regular and repeating pattern of orientation selectivity. We will see later in Section 2.5.13 that convolutional networks and tiled convolutional networks build this repeating structure into models of the visual system, but there

is evidence that our brain does not actually convolve a single function over the visual field, so this must be viewed as a computational trick (Cox and DiCarlo, 2008). Also, convolutional models are typically applied to the standard image coordinates, rather than the warped visual field shown in Figure 2.2. Work such as Larochelle and Hinton (2010) is a step in the direction of using more accurate spatial encoding of V1's retinotopic map.

Physiologically, V1 comprises cells with various computational properties. Many *simple cells* have a firing rate that is well-modelled by a linear filter of a stimulus image (or image sequence), except that they only respond once a certain threshold is met (Carandini and Heeger, 1994; Hubel and Wiesel, 1968, 1962). Simple cells tend to respond when their receptive field looks like an edge at a particular location and orientation, and are often characterized by *Gabor functions*. Consequently, they act like localized complex Fourier transforms. Neurons in V1 that do not follow this linear model are called *complex cells*. Complex cells are sometimes characterized as having orientation selectivity like simple cells, but less sensitivity to the exact retinal location of the change in contrast (phase invariance) (Hubel and Wiesel, 1968, 1962; Movshon et al., 1978). This phase invariance has been defended on theoretical ground (Zetzsche et al., 1999), and incorporated into complex cell models based sums or maximums of simple-cell-like models (Adelson and Bergen, 1985; Riesenhuber and Poggio, 1999), but these models fail to explain a number of other nonlinear effects, such as surround suppression and cross-orientation inhibition (Bonds, 1989; Cavanaugh et al., 2002; Jones et al., 2002). More sophisticated models such as Finn and Ferster (2007); Kouh and Poggio (2008); Rust et al. (2005); Wainwright et al. (2002) include additional parameters to account for these behaviours, but the emerging picture is that V1 neurons seem to resist characterization as functions of images or movies within small receptive fields. For recent reviews of what is known (and unknown) about V1, and how it can be modelled, see Carandini (2006); Chen et al. (2009); Hyvärinen (2009); Olshausen and Field (2005); Simoncelli and Olshausen (2001).

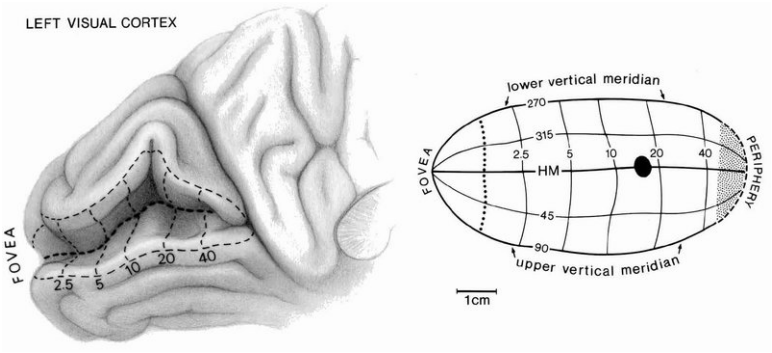

Figure 2.2: Estimate of the representation of the visual field on the surface of human occipital cortex. The figure shows the medial aspect of the occipital lobe with tissue on either side of the calcarine sulcus pushed apart. Right: representation of visual field coordinates as they would appear on the cortex if completely unfolded and flattened. Numbers show either meridional angle or eccentricity in degrees. The dark circle is the blind spot; HM = horizontal meridian. Note the orthogonal intersections of lines of constant eccentricity and meridional angle. From Swindale (2008), which was modified with permission from Horton (2006). See related Figure 2.3 for functional organization across the visual field.

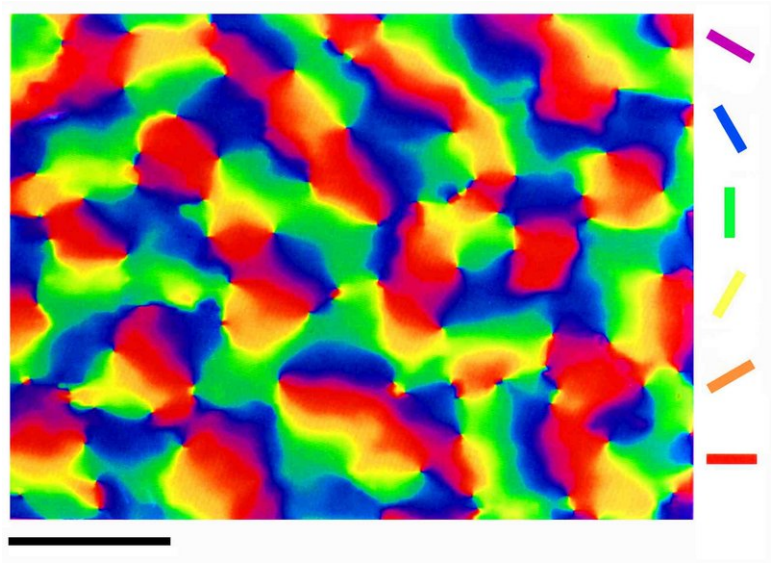

Figure 2.3: Layout of orientation preference observed in area V1 of macaque monkey. Scale bar = 1 mm. From Swindale (2008), modified from Blasdel and Salama (1986).

2.3.5 Beyond Primary Visual Cortex

The ventral stream continues from V1 to areas known as V2, V4, and inferotemporal cortex (IT). The progression from V1 to V2 to V4 and IT is suggested by experiments that measure how much time elapses between the presentation of an image stimulus and the measurement of a neural response. These areas are also retinotopic, but they are smaller than V1. The neurons in these areas have larger receptive fields, are less linear, and are consequently more difficult to characterize than V1 simple cells. Attention plays a larger role in modulating cell firing rates in V2 and V4 than in V1. Some cells in V2 are tuned to certain broader positions, orientations, texture, spatial frequencies and colour, but are also selective to illusory contours (Connor et al., 2007; Gallant et al., 1996; Hegdé and Van Essen, 2000; Kobatake and Tanaka, 1994; Pasupathy and Connor, 2001). There are cells in V4 that are selective to particular simple geometric shapes. In IT we find cells that are highly selective for complex shapes such as faces. IT cortex includes neurons whose firing rates are the *features* we would like in an ideal artificial visual system (Hung et al., 2005; Serre et al., 2007b).

2.3.6 Time, Feedback and Learning in the Visual System

The feed-forward view of the visual system has been very influential on computer vision research, machine learning, and computational neuroscience. My research follows this pattern as well, but some caveats should be mentioned to complement this simplified model.

- The visual system works on a temporal signal, not on images (when studying the pipeline to IT we often ignore the temporal component of the stimulus).
- The cortex of the visual system includes many back-projections–connections that send signals from later stages of processing (according to the feed-forward model) to earlier ones.
- The precise timing of neuron spikes may be important, whereas our models ignore timing and characterize cells only by how frequently they spike.

The speed of our ability to recognize objects speaks to the validity of the feed-forward approximation, but the existence and number of feedback connections reminds us that it is an approximation. Among several problems with the standard feed-forward model, Olshausen and Field (2005) point out that roughly 5% of the excitatory input to layer 4 of V1 comes from the LGN, with the majority of excitation coming from intracortical inputs (Peters et al., 1994), and that somewhere between 60% and 80% of the response of V1 neurons is driven by other V1 neurons, and other inputs than from the retina via the LGN. It has been argued that spike synchrony among subpopulations in visual cortex plays a key role in scene segmentation (Gray, 1999). The feed-forward model makes little room for any sort of scene segmentation, whereas it would seem to be an important property of a good representation for decision making.

Plasticity is the changing of the computational relationship between neurons. Learning would seem to require plasticity, although plasticity might serve other ends as well - such as simply maintaining homeostatis or algorithmic stability in a living brain. Learning in the visual system appears to be continuous, life-long, and fast: for example, IT cells of monkeys that were subjected to artificially modified visual environment adapted to the new environment within 1 hour (Li and DiCarlo, 2008), and just 15 minutes of exposure to a grating pattern can change your visual system for months (McCollough, 1965). The feed-forward rate model does not speak to either the algorithm or the mechanism for learning in the visual system. What learning algorithms we have tend to involve feedback and anatomically we find that indeed there are many axons running *backward* from later stages to earlier ones (in fact even more than carry the signal forward in the first place), but it is not known what algorithm drives learning in the visual cortex (nor in other regions of cortex).

2.3.7 Level of Detail in Computational Modelling

As we look to the brain for inspiration in designing intelligent systems, the question of modelling granularity arises. Are neural spikes necessarily a part of

intelligent functioning, or is it possible to replicate intelligent behaviour with a more approximate neural model? In my work in later chapters, and in most of the machine learning literature, there is a tendency to use rate models (models expressed in terms of the firing rates) instead of spiking ones. The choice is somewhat arbitrary, but not completely. Rate models permit neurophysiological findings to inform us as we think about features, without preventing us from also drawing on many theoretical results and algorithms that inform (a) how those features might be learnt, and (b) what *principles* can we use to choose features for new kinds of stimuli when we do not have neurophysiological hints to help us. Perhaps by identifying such principles we computational people (engineers and modellers) might some day return the favour to the field of neurophysiology as they try to decode representations in new areas of the brain. In the meantime, this level of modelling has led to the engineering of successful models, which match the performance of humans in restricted settings, and advance the state of the art in computer vision.

2.4 Image Features in Computer Vision

This section covers methods for image processing. It describes of some low-level details of the simulations used in later chapters, but it also goes further in the processing pipeline to describe SIFT features (Lowe, 1999). SIFT features are not used in later Chapters because they are designed for large high-resolution images, but they have been a successful and influential approach to image feature extraction in computer vision, and will be discussed in the concluding chapter.

2.4.1 Preprocessing for Images

When presented with a dataset of images $o_i \in \mathcal{O}$, the basic pipeline for converting them to vectors for classification is as follows:

1. standardize colour representation,
2. standardize image size,
3. arrange pixel values in raster order,

4. optionally apply local contrast normalization, and
5. optionally apply one of several affine transforms.

The next few sections describe these steps in more detail.

2.4.1.1 Colour Representation

For eight-bit greyscale images, a single integer in the range [0,255] represents the grey level of a pixel. White corresponds to 255 and black to 0. The relationship between light intensity and these values is roughly logarithmic in terms of energy because our eyes are able to distinguishing differences in light levels mainly relative to one another. Pixel colours can be represented in several formats, which are not linearly related. Some colour representations (such as CMYK) correspond to how one would mix inks to form colour. Other colour representations (such as RGB) correspond to how a cathode ray tube monitor could mix light to form colour. Most of the work in later chapters deals with greyscale images, and where colour images are used colours are encoded in the RGB format, which is the colour encoding that seems to match retinal coding most closely.

2.4.1.2 Resizing and Rasterizing

Images can be stored and represented in many ways on a computer, but for our purposes an image is stored and represented as a matrix of pixel colour values. For an image that is 128 pixels wide and 72 pixels tall, the matrix would have 72 rows and 128 columns. Of course, images gathered from a variety of sources generally come in different shapes and sizes. It is important for classification that such images be adjusted to have a common shape. This adjustment is not done with any sophistication, we simply resize images to have the same numbers of rows and columns, so that they exist in the same vector space. Compared with images registered by the retina, this resizing process ensures that all of the images could at least potentially have been sensed by one eye.

Rasterizing means converting the (2D) matrix of pixel values to a (1D) vector

of those same pixel values. Typically this is done by concatenating the first row of pixels with the second, then the third, and so on. This is simply a rearranging of pixels into an image that is really wide and one pixel tall; no information is lost in this linear conversion. The result of this step is a vector, call it p, of N pixel values p_i. This vector can be used as a feature vector for o directly, or it can be further processed by local contrast normalization and/or affine methods.

2.4.1.3 Local Contrast Normalization

If the result of resizing and rasterizing an image is a vector p of N pixel values p_i, the nonlinear local contrast normalization operator transforms p into a new vector v according to a formula such as

$$v_i = \frac{p_i - \bar{p}}{\sqrt{\frac{1}{N}\sum_j^N (p_j - \bar{p})^2 + \varepsilon}}, \quad \text{where} \quad \bar{p} = \frac{1}{N}\sum_j^N p_j. \tag{2.4}$$

This transform removes the mean pixel value, and makes high-contrast images similar to medium-contrast ones. The ε should be chosen so that low-contrast images remain so (e.g. $\varepsilon = 10$ for grey levels between 0 and 255). This transformation is inspired by the transformation from retinal photoreceptors to retinal ganglion cells.

2.4.1.4 Affine Transformations

If the result of resizing and rasterizing an image (and optionally local contrast normalization) is a vector p of N pixel values p_i, then it may be further processed by an affine transformation. There are several advantages to a well-chosen affine transformation:

- shifting and scaling feature values to occupy the (-1,1) interval improves the ability of optimization methods to find good nonlinear models,
- reducing the full feature set down to a smaller representative subset of features can help optimization methods and also make computations much quicker,
- adjusting the frequency content of images can focus modelling effort on salient

aspects (e.g. edges) of an image.

The simplest technique is to define features x in terms of centred, standardized elements of p.

$$x_i^{(\text{normalized})} = \frac{p_i - \text{mean}_{o \in \mathscr{O}}(p_i)}{\text{Var}_{o \in \mathscr{O}}(p_i)} \tag{2.5}$$

This shifting and scaling helps some optimization algorithms, but does not reduce dimensionality (number of features).

Principal components analysis (PCA) preprocessing refers to the technique of identifying some number M of principal components in the data, and representing an example p by its projections onto those principal components.

$$x^{(\text{pca})} = W\left(p - \text{mean}_{o \in \mathscr{O}}(p)\right) \tag{2.6}$$

Principal components (the columns of W) form an orthogonal basis that spans the directions of greatest covariance in p. Often the columns of W are scaled so that the elements of x have unit variance. PCA preprocessing shifts and scales the features to help optimization algorithms, and also reduces the dimensionality while retaining as much information about the input as possible (for a linear method). This procedure is sometimes called *whitening* because it removes correlation between the elements of x.

ZCA preprocessing is similar to PCA, except that we multiply our features "back" through W to recover features that are interpretable as pixels (see Hyvärinen and Oja, 2000, Section 5.2).

$$x^{(\text{zca})} = W'W\left(p - \text{mean}_{o \in \mathscr{O}}(p)\right) \tag{2.7}$$

Although ZCA does not reduce the dimensionality of the feature vector p, it can smooth out the image and remove subtle high-frequency artifacts that remain after scaling and reshaping.

Sometimes the ZCA procedure is modified slightly to amplify the presence of certain principal components of the data; for example, adding constants to the

$W(p - \text{mean}_{o \in \mathcal{O}}(p))$ term before the second multiplication by W (Coates et al., 2010). The effect is qualitatively similar to band-pass filtering, which is used for example in Olshausen and Field (1996). Band-pass filtering is not optimal in a statistical sense the way PCA and ZCA can be, but it can be performed more quickly by convolutions or a 2-D discrete Fourier transform.

2.4.2 Case of Image Features: SIFT

Later chapters develop models that can be seen as feature extractors from the image representation developed by the previous pipeline. Learning is not necessary though, it is possible to simply program feature extractors with properties such as specificity to high-level shapes and invariance to lighting, translation, rotation, and scaling (e.g. Bosch et al., 2007; Lazebnik et al., 2006; Lowe, 1999). These properties give rise to features that tend to make objects easier to recognize with linear classifiers. One such feature transform is the Scale Invariant Feature Transformation (SIFT: Lowe, 2004). This section looks at the SIFT algorithm to give a sense of how a feature-extraction system can work well without learning.

The SIFT algorithm involves converting an image to *scale space* and then characterizing points in this scale space by image gradient histograms. Scale space of an image is a function $L(x, y, \sigma)$ that is produced by the convolution of a Gaussian filter of width σ with an image $I(x, y)$. The Difference-of-Gaussian image representation is the difference between adjacent levels (defined by multiplicative constant k) in the scale space representation:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.8}$$

The SIFT features characterize an image by describing the extrema (local maxima and minima) in D. There are many local maxima, and it is not possible to identify them all. Fortunately, coarse sampling finds the most salient ones, and Brown and Lowe (2002) describe techniques for refining the coarse estimates.

The SIFT feature is ultimately a normalized, weighted histogram of image gra-

dients in the scale space. Part of SIFT's robustness comes from the fact that those gradients are represented in a canonical reference frame, that is aligned relative to the *overall gradient* in the image in the vicinity of each extremum. The image gradient orientation at each point near an extremum is weighted by the gradient magnitude, up to an experimentally-validated maximum of 0.2. When there are several orientations with near-maximum total weight, a separate set of descriptors is created relative to each of these orientations.

SIFT features are motivated and defended primarily on engineering grounds, but Lowe points out that they are not so far from models of the ventral stream (Lowe, 2004). The simple cell model, especially when it includes lateral inhibition, is consistent with not only the scale space image representation, but also the computation of local extrema in that scale space. Since most simple cells characterize a contrast gradient, it is conceivable that a sum of simple cells with similar orientation selectivity (i.e. a striate column) implements something akin to a counting operation (a histogram bin). Even the normalization of these histogram bin counts seems quite within the realm of cortical computation. The re-orientation of histogram counts into a canonical reference frame is a bit further from standard feed-forward models, but perhaps with simple *dynamic routing* it too is possible in an essentially feed-forward system (Olshausen et al., 1993).

This extends this line of reasoning - if we design a simple model visual system that is simply *capable* of performing these various kinds of computations, can we use automatic methods to tune it for optimal performance? The last background section "Learning Representations" reviews how we can do this with *machine learning* techniques.

2.5 Learning Representations

When we speak of learning representations, we are referring to the approaches of optimization and density modelling, as opposed to the approaches of engineering and neurophysiology.

This section describes the concepts and algorithms used to search function spaces for good features, or for good density models.

2.5.1 Principles for Learning

The most fundamental idea in machine learning is that things about which we learn come from probability distributions, and *learning* means minimizing the expected value of a formally defined *loss function*. This is called *expected risk minimization* (Bottou, 1998; Vapnik, 1989). Any function that maps from a sample of observations to a real value could be considered a loss function. Many loss functions arise in the various applications of machine learning and information retrieval. A loss function simply encodes how satisfied a modeller is with his or her model's performance on particular sample of data. There is hardly a limit to the realm of possible loss functions, but there are two forms of special importance: negative log-likelihood, and zero-one loss.

2.5.1.1 Negative log-likelihood

Negative log-likelihood (NLL, Eq. 2.9) and conditional negative log-likelihood (Eq. 2.10) are often used for density modelling and optimization-based feature extraction approaches, respectively.

$$\mathscr{L}^{(\mathrm{NLL})}(x;\theta) = -\log P_\theta(x) \tag{2.9}$$

$$\mathscr{L}^{(\mathrm{cNLL})}(x,y;\theta) = -\log P_\theta(y|x) \tag{2.10}$$

In these equations x denotes a stimulus (which will typically be an image in later chapters), and y denotes the correct label (among a finite set of possibilities) associated with that stimulus. The θ denotes a model that tells us how to evaluate the probabilities on the right hand sides of Equations 2.9 and 2.10.

In *learning by maximum likelihood*, we seek to minimize the expectation

$$\mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}^{(\mathrm{NLL})}(x;\theta)] \tag{2.11}$$

with respect to θ, where x is distributed according to its natural distribution. [1] In learning by conditional maximum likelihood we seek to minimize the expectation

$$\mathbb{E}_{(x,y)\sim\mathscr{G}_{x,y}}[\mathscr{L}^{(\text{cNLL})}(x,y;\theta)]. \tag{2.12}$$

2.5.1.2 Zero-One Loss

In classification settings we often do not care about probabilities *per se*. We see our model θ not as a probability distribution, but as a method for making a label prediction $f_\theta(x)$ for some stimulus x. We want those predictions to be correct as often as possible, and the zero-one loss function reflects this desire.

$$\mathscr{L}^{(01)}(x,y;\theta) = \mathscr{I}_{y\neq f_\theta(x)} \tag{2.13}$$

The zero-one loss is named for the fact it is 0 when the prediction is correct and 1 otherwise. If we could minimize the expectation

$$\mathbb{E}_{(x,y)\sim\mathscr{G}_{x,y}}[\mathscr{L}^{(01)}(x,y;\theta)] = \mathbb{E}_{(x,y)\sim\mathscr{G}_{x,y}}[\mathbb{I}_{y\neq f_\theta(x)}] \tag{2.14}$$

we would be minimizing the rate of incorrect predictions in the domain $\mathscr{G}_{x,y}$ of interest.

2.5.1.3 Regularization

Often we augment a loss function with a *regularization term* $R(\theta)$ that does not depend on x or y. For example regularized maximum likelihood learning could be written as

$$\mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}^{(\text{NLL})}(x;\theta)] - R(\theta) \tag{2.15}$$

1. I'll use the term natural distribution to describe the distribution that nature imposes on some observed quantity, to contrast with variables that follow formalized distributions. Other authors have called this distribution the grand truth (Bottou, 1998) or ground truth distributions.

This term allows the modeller to express a preference for various models θ, without regard to their compatibility with x and y. If we view the optimization of this expectation with respect to θ as Bayesian maximum a-posteriori inference, then this extra term expresses our prior over models.

2.5.2 Training, Testing, and Cross-Validation

It was mentioned in the previous section that learning means minimizing the expected value of some formally defined loss function $\mathscr{L}$. In this section we develop this concept in more detail.

Suppose our goal in learning is to minimize, for some $\mathscr{L}$ and natural distribution $\mathscr{G}_x$, the expectation

$$\mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}(x;\theta)]. \tag{2.16}$$

Generally we have at our disposal only a finite sample $\mathscr{X}$ whose elements $x \in \mathscr{X}$ have been drawn identically and independently, $x \sim \mathscr{G}_x$. *Testing* means using $\mathscr{X}$ (typically a subset) to estimate the expected loss incurred by a particular predictor f_θ. *Training* a model means minimizing some estimator of this expectation with respect to θ.

2.5.2.1 Testing

Testing is relatively straightforward. It is done by simply replacing the expectation with an empirical mean over some subset $\mathscr{X}^{(\text{test})} \subset \mathscr{X}$,

$$\mathbb{E}x \sim \mathscr{G}_x[\mathscr{L}(x;\theta)] \approx \frac{1}{|\mathscr{X}^{(\text{test})}|}\sum_{x\in\mathscr{X}^{(\text{test})}} \mathscr{L}(x;\theta). \tag{2.17}$$

It is important for the approximation that $\mathscr{X}^{(\text{test})}$ be representative of $\mathscr{G}_x$, in practice this typically means that no element $x \in \mathscr{X}^{(\text{test})}$ should have been used by the training procedure that selected the optimal value for θ.

2.5.2.2 Training

Given dataset $\mathscr{X}$ and test set $\mathscr{X}^{(\text{test})}$, define a *development set* $\mathscr{X}^{(\text{dev})} = \mathscr{X} / \mathscr{X}^{(\text{test})}$ as the complement of $\mathscr{X}^{(\text{test})}$ within $\mathscr{X}$. Training the model is the rather open-ended optimization problem

$$\theta^* = \arg\min_{\theta \in \Theta} \frac{1}{|\mathscr{X}^{(\text{dev})}|} \sum_{x \in (\mathscr{X}^{(\text{dev})})} \mathscr{L}(x; \theta) \tag{2.18}$$

of finding the best model θ among a set Θ of possible models. There are as many approaches to this problem as there are optimization methods. One in particular deserves mention as being particularly general and useful: it is the method of training by what is called *cross-validation*. Even when other training algorithms are used, they are almost always used within a cross-validation algorithm.

2.5.2.3 Cross-validation

Cross-validation refers to the technique of choosing the best-performing element of a finite set of candidate models $\Theta_K = \{\theta_1, \theta_2, ...\theta_K : \theta_i \in \Theta\}$ after literally trying each one on fake test examples called *validation set*. We choose a validation set $\mathscr{X}^{(\text{valid})}$ from among the training examples $\mathscr{X}^{(\text{valid})} \subset \mathscr{X}^{(\text{dev})}$ so that it does not overlap with $\mathscr{X}^{(\text{test})}$. Then we choose the best model by simulating the test scenario using our validation data:

$$\theta^* = \arg\min_{\theta \in \Theta_K} \frac{1}{|\mathscr{X}^{(\text{valid})}|} \sum_{x \in \mathscr{X}^{(\text{valid})}} \mathscr{L}(x; \theta). \tag{2.19}$$

Cross-validation is always the outermost training algorithm in the work presented in later chapters, although it is usually not even thought of as a "training algorithm" because it is so simple. Technically though, choosing a model by cross-validation is a perfectly valid training algorithm.

Usually what is called a "training algorithm" is a procedure for searching an infinite model space, and recovering a model that will serve as one of the $\theta_i \in \Theta_K$

in the cross-validation procedure. Such a procedure also typically requires data, which is called the *training set*, $\mathscr{X}^{(\text{train})}$. In order for the cross-validation procedure to give an unbiased estimate of test set performance, it is typically necessary that $\mathscr{X}^{(\text{train})} \subseteq \mathscr{X}^{(\text{dev})} / \mathscr{X}^{(\text{valid})}$.

2.5.2.4 Overfitting

Overfitting occurs when we optimize over a set Θ that is too large relative to the size of the dataset $\mathscr{X}^{(\text{dev})}$ or $\mathscr{X}^{(\text{train})}$ used to constrain the optimization procedure. An example will illustrate the phenomenon. Suppose we were trying to classify real numbers with either the label 'A' or 'B'. It is natural in this setting to minimize a Zero-One loss. Suppose we were going to do learning by cross-validation and we had four models in Θ_K:

1. model θ_1 that always returns label 'A',
2. model θ_2 that always returns label 'B',
3. model θ_3 that returns 'A' for negative x and 'B' for non-negative x, and
4. model θ_4 that returns 'B' for negative x and 'A' for non-negative x.

Suppose the natural distribution here were that x could take values -1 or 1 with equal probability, and that if it were 1 it tended to be labelled 'A' and if it were -1 it tended to be labelled 'B'. The natural distribution is noisy, so none of the models is perfect, but θ_4 is our best choice in terms of minimizing the Zero-One loss.

To see the danger of overfitting, suppose furthermore that our validation set $\mathscr{X}^{(\text{valid})}$ had only a single example in it. No matter what value our validation example were to have, or how it were to be labelled, it could only allow us to reject two of the four candidate models. The other two models would appear to have perfect accuracy on the validation set. Since the dataset used to choose among models (in this case $\mathscr{X}^{(\text{valid})}$) is not sufficiently large to narrow down the options in the model family (here Θ_K) to a single choice, we say that model family Θ *overfits* the dataset. The consequence of overfitting is that success on the training data or

validation data does not carry over, or *generalize*, to the test data. The *capacity* of a family $\mathbf{\Theta}$ is related to the minimum size of dataset for which overfitting can be a problem. Capacity can be difficult to measure, but there are techniques such as ones based on VC-dimensionality for models that are classifiers (Abu-Mostafa, 1989; Vapnik, 1989; Vapnik and Chervonenkis, 1971; Vapnik et al., 1994). Since we often use model families $\mathbf{\Theta}$ that contain infinitely many models, and which contain models that are arbitrarily close to any smooth bounded function, overfitting is often an issue in practice, and finding models that generalize to test data is of primary interest.

2.5.2.5 Inductive Bias

Closely related to the phenomenon of overfitting is the concept of an inductive bias. Returning to the previous example, our lone validation example necessarily provided evidence against two of the models and in favour of the other two models. Inductive bias refers to whatever principle or heuristic we use to choose a best model after considering that evidence. Inductive bias is the result of using a regularization term along with a loss function. In a case that is so extremely under-specified by the data the inductive bias plays a central role in choosing the best model; when the evidence is stronger the inductive bias is less relevant. Note that a second example, and generally more data would have provided more evidence one way or the other for each model, but ultimately in a regularized setting, a training procedure can always return a model that is not the one that realizes the best likelihood or Zero-One score. The notion of an inductive bias is especially important in settings where the model family $\mathbf{\Theta}$ under consideration can approximate *any* continuous smooth function arbitrarily well. Such families can theoretically overfit any dataset, so regularization that implements and reflects a good inductive bias is crucial for finding a model that generalizes.

2.5.3 Learning Linear Classifiers

Recall the *linear classifier* introduced in Section 2.1.

$$\arg\max_{l \leq L} (Wx + b)_l \tag{2.20}$$

In this section we will look at two ways to learn an optimal linear classifier from data. The first is based on an algorithm called the Perceptron algorithm. The second is based on a standard statistical formalism called logistic regression.

2.5.3.1 Perceptron, Hinge Loss

One loss function we can use to train a linear classifier is the following, called a *hinge loss* (Cortes and Vapnik, 1995). Supposing we had some feature vector x and true label y, $1 \leq y \leq L$, we define l^* to be the most likely incorrect label, and then we can define the hinge loss function.

$$l^* = \arg\max_{l \leq L, l \neq y} (Wx + b)_l \tag{2.21}$$

$$\mathscr{L}^{(hinge)}(x, y; \theta) = \max(0, \varepsilon - (Wx + b)_y + (Wx + b)_{l^*}) \tag{2.22}$$

The hinge loss function demands that the correct label win in the argmax by at least some margin ε over the runner-up. In the original Perceptron model and learning algorithm $\varepsilon = 0$ was used, and the algorithm was only described for two classes. Subsequently it has been found that it is better to require the correct class to win by a more sizable margin over the next-best class. I prefer this particular generalization from two classes to many classes, which has been suggested by Bordes et al. (2007) although other definitions of the margin in multiclass settings have been suggested as well (Allwein et al., 2000).

To find the best $(W, b) = \theta$ for a particular dataset $\mathscr{X}$, we must solve the

following optimization problem:

$$\arg\min_{\theta} \text{mean}_{(x,y)\in\mathscr{X}} \left(\mathscr{L}^{(hinge)}(x,y;\theta) \right). \tag{2.23}$$

Any one of a number of gradient-based optimization methods can be brought to bear on this problem. There are several points of non-differentiability in this system: it is non-differentiable when two incorrect labels tie in the argmax, and it is non-differentiable when the margin is exactly ε. Typically neither of these situations arise near the optimum value of θ so it is safe to assign a derivative of zero at such points for the purpose of doing numerical optimization.

2.5.3.2 Boltzmann, Softmax, Logistic Regression

While the hinge loss method is often the best choice when we are only interested in a prediction, sometimes we are also interested in algorithms that provide a conditional probability distribution over all the class labels.

The Boltzmann distribution is a common choice:

$$P_\theta(y=l|x) = \frac{e^{(Wx+b)_l}}{\sum_k e^{(Wx+b)_k}} \tag{2.24}$$

This distribution is tightly related to the *softmax* function. The softmax function from $\mathbb{R}^L \to \mathbb{R}^L$ is the one that maps in this case from the vector $Wx+b$ to the vector whose l^{th} element is $P_\theta(y=l|x)$. So, in terms of the softmax function, we can rewrite P_θ more compactly as

$$P_\theta(y=l|x) = \text{softmax}(Wx+b)_l \tag{2.25}$$

Our purpose here in defining $P_\theta(y=l|x)$ is to bring the conditional negative log-likelihood loss function to bear on the problem. To find the best $(W,b)=\theta$ for a particular dataset $\mathscr{X}$ (i.e., to perform maximum likelihood learning) we must

solve the following optimization problem:

$$\arg\min_{\theta} \quad \text{mean}_{(x,y)\in\mathscr{X}} \left(\mathscr{L}^{(cNLL)}(x, y; \theta) \right) \tag{2.26}$$

$$= \arg\min_{\theta} \quad \text{mean}_{(x,y)\in\mathscr{X}} \left(-\log P_\theta(y|x) \right) \tag{2.27}$$

$$= \arg\min_{\theta} \quad \text{mean}_{(x,y)\in\mathscr{X}} \left(\log \left(\sum_{k=1}^{L} e^{(Wx+b)_k} \right) - (Wx+b)_y \right) \tag{2.28}$$

Again, any one of a number of gradient-based optimization methods can be brought to bear on this problem. Unlike the system that resulted from the hinge loss, this system is differentiable everywhere.

2.5.4 Learning by Gradient Descent

Any number of gradient-based optimization algorithms could potentially be used to train the linear classifiers as described in Sections 2.5.3.1 and 2.5.3.2, or the non-linear models and classifiers discussed in later sections. This section describes three increasingly complex gradient descent algorithms, the culmination of which is annealed stochastic gradient descent with early stopping. This last algorithm is employed in the subsequent chapters of this thesis to fit both supervised and unsupervised models.

2.5.4.1 Batch Gradient Descent

Batch gradient descent (Algorithm 1) is the simplest gradient-based minimization algorithm. The algorithm is to repeatedly move parameters by some small amount λ in whichever direction most rapidly reduces the mean loss $\mathscr{L}$. Every iteration over the dataset (the **for all** loop) is called an *epoch* of training.

The batch gradient descent algorithm itself specifies neither the starting conditions (how to initialize θ) nor the stopping condition. Algorithms typically initialize θ to small random values, although we will see in Section 2.5.11 (on pretraining) that there are important exceptions to this general rule. There are several heuris-

Algorithm 1 Batch gradient descent

$\theta_j \leftarrow$ some initial value $\forall j$
while termination condition not satisfied **do**
  $g_j \leftarrow 0$
  epoch $\leftarrow 0$
  for all $(x,y) \in \mathscr{X}$ **do**
    $g_j \leftarrow g_j + \frac{1}{|\mathscr{X}|} \frac{\partial \mathscr{L}(x,y;\theta)}{\partial \theta_j}, \ \forall j$
  end for
  epoch $\leftarrow 1+$ epoch
  $\theta_j \leftarrow \theta_j - \lambda_j g_j, \ \forall j$
end while

tic stopping conditions in common use, including simply stopping when the epoch exceeds a predetermined threshold. To describe the most widely used kind of stopping condition though, we will need to return to the cross-validation algorithm and make it a little more complicated.

2.5.4.2 Batch Gradient Descent with Early Stopping

The most widely used stopping conditions are called early stopping conditions. They are termination conditions based on validation set performance. To describe an early stopping algorithm we must distinguish between the datasets used for training ($\mathscr{X}^{(\text{train})}$) and validation ($\mathscr{X}^{(\text{valid})}$), and also between the loss function $\mathscr{L}^{(\text{true})}$ that we really care to minimize and the *surrogate* loss function $\mathscr{L}^{(\text{surr})}$ that is differentiable, but otherwise similar in shape to the true loss. We distinguish between these two loss functions when learning to classify because the misclassification rate $\mathscr{L}^{(01)}$ is not a differentiable function.

Algorithm 2 describes the general form of gradient descent with early stopping, with a placeholder function called terminate() that decides whether to stop training based on the current epoch, the current score, the best epoch seen so far, and its score.

Algorithm 2 Batch gradient descent with early stopping

$\theta_j \leftarrow$ some initial value $\forall j$
epoch $\leftarrow -1$
score $\leftarrow \infty$
best epoch $\leftarrow$ -1
best score $\leftarrow \infty$
best $\theta \leftarrow$
while not terminate(epoch, score, best epoch, best score) **do**
  epoch $\leftarrow 1+$ epoch
  $g_j \leftarrow \frac{\partial \text{mean}_{(x,y)\in\mathscr{X}^{(\text{train})}} \mathscr{L}^{(\text{surr})}(x,y;\theta)}{\partial \theta_j}, \forall j$
  score $\leftarrow \text{mean}_{(x,y)\in\mathscr{X}^{(\text{valid})}} \mathscr{L}^{(\text{true})}(x,y;\theta)$
  if score $<$ best score **then**
    best epoch $\leftarrow$ epoch
    best score $\leftarrow$ score
    best $\theta \leftarrow \theta$
  end if
  $\theta_j \leftarrow \theta_j - \lambda_j g_j, \forall j$
end while

2.5.4.3 Annealed Stochastic Gradient Descent with Early Stopping

The algorithm of batch gradient descent with early stopping is useful as an illustration of how to combine early stopping with gradient descent, but computing the gradient for each parameter from the entire training set is computationally expensive and unnecessary. If you do use the entire training set to estimate the gradient, then there are more sophisticated optimization strategies such as conjugate gradient methods (Ueberhuber, 1997) that are nearly always better than this algorithm. Such methods will make steps in different directions than the g_j, and typically converge to a minimum in fewer epochs.

However, in learning applications it is typically faster yet to appeal to stochastic optimization methods. Since we are trying to minimize an expected loss function, we can get unbiased estimates of the gradient by taking the average across any number of samples from the training set - we do not need to use the entire training set to estimate the gradient. Especially when there are many training examples, we can often get quite good estimates of the gradient from a tiny fraction of the

training set, called a *mini-batch*. Stochastic gradient descent with early stopping and an annealed learning rate (Algorithm 3) is a simple but effective learning algorithm in many linear and non-linear machine learning applications, and it is used throughout the experiments in later chapters.

Algorithm 3 Annealed stochastic gradient descent with early stopping

$\theta_j \leftarrow$ some initial value $\forall j$
iter $\leftarrow -1$
score $\leftarrow \infty$
best iter $\leftarrow -1$
best score $\leftarrow \infty$
best $\theta \leftarrow$
while not terminate(iter, score, best iter, best score) **do**
  sample $\mathscr{X}^{(\text{mini})} \subset \mathscr{X}^{(\text{train})}$
  $\theta_j \leftarrow \theta_j - \frac{\lambda_j}{\max(1, t/\tau)} \frac{\partial}{\partial \theta_j} \text{mean}_{(x,y) \in \mathscr{X}^{(\text{mini})}} \mathscr{L}^{(\text{surr})}(x, y; \theta), \ \forall j$
  $t \leftarrow t + 1$
  if validate(iter) **then**
    score $\leftarrow \text{mean}_{(x,y) \in \mathscr{X}^{(\text{valid})}} \ \mathscr{L}^{(\text{true})}(x, y; \theta)$
    if score $<$ best score **then**
      best epoch $\leftarrow$ epoch
      best score $\leftarrow$ score
      best $\theta \leftarrow \theta$
    end if
  end if
end while

Annealing refers to the use of $\frac{\lambda_j}{\max(1, t/\tau)}$ instead of λ_j as the learning rate. As long as $t \leq \tau$ this has no effect, but as t increases beyond τ the effective learning rate decreases gradually to 0. The reason we anneal the learning rate is to guarantee convergence to a minimum. As the learning rate approaches 0, the parameters changes less and less on every iteration, so the direction of descent approaches the direction of the batch learning algorithm. No matter what the variance in our noisy estimates of the gradient, the annealed algorithm will converge to a local minimum of the surrogate loss function on the training set.

Algorithm 3 is still a template in the sense that it does not specify how θ should be initialized, when to terminate, or when to validate. I usually initialize θ to

small random values (e.g. uniformly in a range such as -0.1 to 0.1), although some researchers argue that the initial range should be scaled according to architectural properties of a particular model (Bengio and Glorot, 2010; LeCun et al., 1998b). It is common to choose to terminate when the epoch exceeds the best epoch by some sufficient amount. My favourite early stopping recipe is to wait for at least a few iterations through the training set (maybe 10 or 50, although it really depends on the size of the training set and difficulty of the problem), and thereafter to stop whenever the epoch is twice the best epoch. I usually use a validation set of ten thousand examples, and check validation performance every fifty thousand examples.

2.5.5 Regularization in Linear Classifiers

Recall that regularization (Section 2.5.1.3) is used to indicate a prior preference over models that will be found by the learning process. The two most common regularization methods used with linear classifiers are called ℓ_1 and ℓ_2 regularization. In ℓ_1 regularization, we add a penalty $R^{(1)}(\theta) = \gamma_1 \sum_{ij} |W_{ij}|$ to the loss function. In ℓ_2 regularization, we add a penalty $R^{(2)}(\theta) = \gamma_2 \sum_{ij} W_{ij}^2$ to the loss function. The effect of ℓ_1 regularization is to sparsify W. If there are many features and we believe that a large fraction of them can safely be ignored, then a positive γ_1 during training can improve generalization. The effect of ℓ_2 regularization is to shrink elements of W in proportion to their length, so it has almost the opposite effect to ℓ_1. If many features are noisy and redundant this regularization can improve generalization by encouraging the model to use them all in equal measure.

2.5.6 Hyper-parameters

Ideally, a learning algorithm for model family Θ should be seen as a function that maps a dataset $\mathscr{X}$ to an optimal model $\theta \in \Theta$ with minimal requirements in terms of computational and spatial complexity.

In practice though, the situation is more complicated. Taking our linear classi-

fiers as an example, we saw in Section 2.5.3.1 and Section 2.5.3.2 that there are two ways to train a linear classifier: by hinge loss, and by negative log likelihood. In each method there is a learning rate, and potentially distinct learning rates for the W term and the b term. There is also the time constant τ that governs the annealing schedule. If we consider regularization strategies such as ℓ_1 or ℓ_2 penalization of W, then we must choose coefficients γ_1 and γ_2. Stochastic gradient descent cannot help us to choose these constants, so they are called *hyper-parameters*. Hyper-parameters must generally be selected by cross-validation. There is no shortcut - in practice we must select several hyper-parameter assignments, and run SGD for each one to see what validation score it yields. Our ultimate learning algorithm is to return the model with the best validation score.

How do we select which hyper-parameter assignments to try? Typically we experiment with settings by hand for a while, and then perform a grid search to explore the most promising possibilities more systematically. I will argue in Chapter 7 that simple random searches are a better way than grid search to choose assignments.

2.5.7 Feature Learning by Optimization - Neural Networks

Neural networks for pattern classification can be seen as an extension to the linear classifiers seen in Sections 2.5.3.1 and 2.5.3.2 (LeCun, 1985; Rumelhart et al., 1986b). Instead of applying linear function W to the feature vector, neural networks apply a non-linear function instead. Recalling the *linear classifier* introduced in Section 2.1, $l^* = \arg\max_{l \leq L} (Wx+b)_l$, a one-hidden-layer tanh neural network would be

$$l^* = \arg\max_{l \leq L} \left(W \tanh(Vx+c)+b\right)_l, \tag{2.29}$$

where $V \in \mathbb{R}^{K \times N}$ is called the input weight matrix, and $c \in \mathbb{R}^K$ is called the hidden unit bias. The *hidden representation* is the vector of activities that was substituted for x in the original linear model (here, $\tanh(Vx+c)$). The elements of this vector are called *hidden units*. The parameter pair (V,c) defines what is called a *layer* of a

neural network, because they can be stacked to form *deep networks*. For example, we can extend our tanh neural network to have two hidden layers by adding another parameter pair (U,d) similarly to how we added V,c to the original linear model:

$$l^* = \arg\max_{l \leq L} \left(W \tanh(V \tanh(Ux+d)+c)+b\right)_l, \tag{2.30}$$

In deep networks it is customary to refer to the tanh outputs of all layers as hidden units, regardless of whether they are used directly by the linear classifier (W,b).

It is also common to use other non-linear *activation functions* in place of the tanh. The most common alternative in older literature is the [logistic] sigmoid $\sigma(z) = 1/(1+e^{-z})$, but some more recent literature uses a hard tanh $\max(-1, \min(z,1))$, or a rectified linear unit $\max(0,z)$ (Bengio and Glorot, 2010; Collobert and Weston, 2008; LeCun et al., 1998b; Nair and Hinton, 2010; Rumelhart et al., 1986a). Later chapters in this dissertation examine the utility of activations that are inspired more or less by neurophysiological characterizations of complex cells in visual area V1.

The reason to use a neural network instead of a linear model is to draw on the *optimization approach* to feature learning to find better features. With neural network models, we use a gradient-based training method (such as the SGD algorithm described in Section 2.5.4.3), to optimize not only W and b, but in fact all of the parameters $\theta = (W,b,V,c,U,d,...)$. As a modeller, all that is necessary is to choose the mathematical form of a good feature, and SGD will choose parameter values that optimize training set performance.

There are two drawbacks to this approach: firstly, with sufficient model capacity it is often trivial for SGD to minimize $\mathscr{L}^{(\text{surr})}$ on the training set so regularization becomes too important relative to the fitting of training data; secondly, with insufficient model capacity SGD often fails to find good solutions that exist in Θ because the training loss function is not convex and has many local minima.

The first problem is a straightforward case of overfitting. A neural network with a single tanh output with even a single layer of (sufficiently many) hidden units can

approximate any smooth function on the (-1,1) range to arbitrary precision (Hartman et al., 1990; Hornik, 1989). Regularization of these models is thus essential. In early neural network literature it was believed that the most important method of regularization was careful tuning of the size (K) of hidden layers. That practice has given way in more modern work to regularization by early stopping. With SGD, early stopping implements a kind of ℓ_2 regularization implicitly when parameters are initialized close to zero in the sense that solutions close to the origin are found more easily.

The second problem is that neural networks present a difficult optimization problem. Current research is aimed at overcoming this by better initialization, and better optimization algorithms than SGD. The following sections on Support Vector Machines and Restricted Boltzmann Machines will look at two approaches to better initialization, in preparation for the Spike and Slab Restricted Boltzmann Machine presented in Chapter 5. There are several algorithms that leverage the techniques of second-order batch gradient descent methods in the setting of stochastic optimization, but I do not use them in the work presented in later chapters (Bordes et al., 2009; Le Roux et al., 2008; Martens, 2010; Schraudolph, 2002; Schraudolph and Graepel, 2003; Schraudolph et al., 2007).

2.5.8 Support Vector Machine

Support Vector Machines (SVMs) have the form of single-layer neural networks, but they use kernel-based features instead of direct features (Cortes and Vapnik, 1995; Vapnik, 1982). There is a large literature on kernels for various kinds of data, but in later chapters two kinds of kernel are used: the Gaussian kernel and the polynomial kernel. These kernel functions apply to pairs (a, b) from the dataset

$\mathscr{X}$:

$$k^{(\text{gauss})}(a,b;\gamma) = e^{-\gamma||a-b||^2} \tag{2.31}$$

$$k^{(\text{poly})}(a,b;\gamma,\delta,\varepsilon) = (\gamma a \cdot b + \delta)^{\varepsilon} \tag{2.32}$$

$$\tag{2.33}$$

In a support vector machine, the *hidden representation* of an example a is the hidden vector whose elements h_i are the kernel values when a is paired with every element of the training set. The classification model given some kernel choice k is

$$l^* = \underset{l \le L}{\arg\max} \, (Wh + b)_l, \text{ where } h_i = k(a, \mathscr{X}_i^{(\text{train})}). \tag{2.34}$$

It is theoretically possible to use optimization methods to train this model like a neural network, but typically kernel functions are chosen to have very few parameters and it would nullify the principal advantage of using an SVM, namely that the only parameters to train are W and b. Of course, training W and b is typically more expensive than it would be in a conventional neural network, because the hidden representation h of the entire training set has $\mathscr{O}(|\mathscr{X}^{(\text{train})}|^2)$ elements.

The challenge in using a kernel and SVM approach is to generalize beyond the training data. For example, the Gaussian kernel SVM (Gaussian SVM) can be seen as an interpolation scheme that smooths out the training data in order to minimize the number of classification errors. Gaussian SVMs are theoretically limited in how efficiently they can generalize from data (Bengio et al., 2006). In the context of images for example, the Gaussian kernel (and polynomial kernel) similarity between a white line on a black background and the same white line shifted over by just one pixel is equal to the similarity with any other image not overlapping the original white line. These kernels are not able to recognize two images as being the same *but for some translation*, or the same *but for a rotation*. This problem is not limited to Gaussian kernels, it is present in a broad class of functions called *shallow models* (Bengio, 2009; Bengio and LeCun, 2007). In

order to recognize similarities such as the translated white line, we must look to *deep models* that use several stages of processing to turn images into features. Stages of processing could mean applying non-linear filters, collecting statistics at automatically-determined interest points, and even trying to infer 3D shape from image fragments to recognize familiar objects in novel poses. The design of more exotic domain-specific kernels is a topic of current research, but typically this is done by leveraging domain-specific direct features, such as the SIFT feature we saw in Section 2.4. Similarly, advances in direct feature learning, as opposed to direct feature engineering, can also be viewed as advances in the design of more sophisticated kernels.

2.5.9 Gaussian Mixture Models

A Gaussian mixture model (GMM, also mixture of Gaussians) is a probability density over a Euclidean space. It is parametrized by some number $K \in \mathbb{N}$ of means $\mu_i \in \mathbb{R}^N$, variances $\sigma_i^2 \in \mathbb{R}$, and priors $\phi_i \in \mathbb{R}$ such that $\phi_i \geq 0$ and $\sum_i \phi_i = 1$. A GMM model is parametrized by $\theta = (\mu, \sigma, \phi)$. It is a latent variable model, with latent variable in Z, where Z is the natural basis for $\mathbb{R}^K$: The element $Z_i \in Z$ is zero everywhere except for a value of 1, in the $i'th$ dimension.

The density model over $\mathbb{R}^N$ factors is

$$p^{(\mathrm{GMM})}(x;\theta) = \sum_i p(x|Z_i;\theta)P(Z_i;\theta) \tag{2.35}$$

$$= \sum_i \frac{\phi_i e^{-\frac{||x-\mu_i||^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i^N}. \tag{2.36}$$

If we are interested in the posterior probability over Z given x, we can apply

Bayes rule to see that

$$p^{(\text{GMM})}(z = Z_i|x,\theta) = \frac{p(x|z = Z_i,\theta)p(z = Z_i|\theta)}{p(x|\theta)} \tag{2.37}$$

$$\propto p(x|z = Z_i,\theta)p(z = Z_i|\theta) \tag{2.38}$$

$$= \frac{\phi_i}{\sqrt{2}\sigma_i^N} e^{-\frac{||x-\mu_i||^2}{2\sigma_i^2}} \tag{2.39}$$

$$= \zeta_i e^{-\gamma_i||x-\mu_i||^2}. \tag{2.40}$$

2.5.10 SVM and Latent Variable Classifiers

The Parzen window density model is a special form of GMM where we take $K = |\mathscr{X}|$, all the $\phi_i = \frac{1}{K}$, all $\sigma_i^2 = \sigma^2$, and $mu_i = \mathscr{X}_i$. With this done, the latent posterior distribution $P^{(\text{Parzen})}(z = Z_i; x, \theta)$ is proportional to $e^{-\gamma_i||x-\mu_i||^2}$. If we simply fold the constant of proportionality into W, we can see that a Gaussian SVM is a linear classifier applied to the vector h whose elements h_i are defined by the posterior distribution $P(z = Z_i|x,\theta)$ of the latent variable z in a Parzen window density model of the training data. The connection between features and latent variables of density models has inspired much of the algorithmic progress in *deep learning* seen in recent years (Bengio and LeCun, 2007; Dahl et al., 2010; Erhan et al., 2010; Hinton et al., 2006; Lee et al., 2009; Raina et al., 2007). Just as the Parzen windows model could be used as a better initial value for a one-layer neural network with a radial basis activation function (to form a Gaussian SVM) so can other models provide better initial values for one- or many-layered neural networks with a variety of activation functions. The technique of initializing a neural network so that its hidden representation is the posterior probability of a latent variable in some density model is known as *pretraining* the network using *unsupervised learning*.

Design by density modelling is the approach of using latent (unobserved) variables of a probability model of the image/stimulus as features for classification. The success of this approach depends on the design of the probability model. Successful models typically have latent variables that correspond to *causes* of the whole

image or part of it. We can think of an image as being *caused* by the properties of the objects that appear, so when the classification task is related to the properties of those images, we can expect the latent variables to be good features. Density modelling approaches have become increasingly popular, especially as several density modelling approaches have corroborated findings in neurophysiology that V1 simple cells form a bank of Gabor-like edge detectors such as Sparse Coding (Lee et al., 2008a; Mairal et al., 2010; Olshausen and Field, 1996; Ranzato et al., 2008; Rao and Ballard, 1999), Independent Components Analysis (Hyvärinen and Hoyer, 2001; Hyvärinen et al., 2001), Slow Feature Analysis (Berkes and Wiskott, 2005; Berkes et al., 2009a; Sprekeler et al., 2007), Restricted Boltzmann Machine (Hinton, 2007), Denoising Auto-Encoders (Vincent et al., 2010). Although the design of a probability model involves a substantial dose of intuition, there are often effective generic algorithms for fitting them to data. Several of these models and the algorithms that fit them to data will be described in the following sections.

2.5.11 Restricted Boltzmann Machines

The argument that neural networks should be pretrained by unsupervised learning was advanced with the use of the Restricted Boltzmann Machine (Bengio, 2009; Freund and Haussler, 1994; Hinton, 2002; Smolensky, 1986). The Restricted Boltzmann Machine (RBM) is an *energy-based model* because it is defined as a joint distribution based on an *energy function*. The RBM is a model of two binary valued vectors $v \in \mathbb{B}^N$ and $h \in \mathbb{B}^K$. It is parametrized by a *weight matrix* $V \in \mathbb{R}^{N \times K}$ and two *bias vectors* $a \in \mathbb{R}^N$ and $b \in \mathbb{R}^K$. The energy function $E(x,h)$ is

$$E(x,h) = -a'x - h'Vx - b'h \tag{2.41}$$

In energy-based models, an energy function is related to a probability distribution by the exponential function

$$P_\theta^{(\mathrm{RBM})}(x,h) = \frac{1}{Z(\theta)} e^{-E(x,h;\theta)}. \tag{2.42}$$

The constant of proportionality $Z(\theta)$ is the reciprocal of what is known as the *partition function*. The partition function is usually written as Z, but is not to be confused with the latent variable in the GMM described in Section 2.5.9.

In RBMs, it is interesting to consider the forms of the conditional probabilities of x and h, when conditioning on the value of the other:

$$P^{(\text{RBM})}(h_i = 1|x) = \sigma(Vx + b), \text{ and} \tag{2.43}$$

$$P^{(\text{RBM})}(x_i = 1|h) = \sigma(V^T h + c), \tag{2.44}$$

where σ is the logistic sigmoid activation function. The inference of $P(h_i = 1|x)$ corresponds exactly to the activation function in the hidden layer of a sigmoid-based neural network. Consequently, with a trained RBM we could initialize a sigmoidal neural network from an RBM analogously to how we can initialize a radial-basis neural network from a Parzen windows model.

RBMs can be trained by an algorithm called Contrastive Divergence (CD, Hinton, 2002). A special algorithm is required because general gradient-based methods require that we compute the gradient of the probability of data with respect to the partition function. The partition function of an RBM involves a summation over all possible binary vector values for either x or h, of which there are 2^N and 2^K respectively. This computation is intractable in models where both N and K are larger than around 16, and therefore intractable in models that are big enough to be useful.

2.5.11.1 Contrastive Divergence

The maximum likelihood gradient on the parameters of an energy model have a general form:

$$\frac{\partial \mathbb{E}_{x\sim\mathscr{G}_x}[-\log P_\theta(x)]}{\partial \theta} = \mathbb{E}_{x\sim P_\theta}\left[\frac{\partial E(x)}{\partial \theta_i}\right] - \mathbb{E}_{x\sim\mathscr{G}_x}\left[\frac{\partial E(x)}{\partial \theta_i}\right]. \tag{2.45}$$

In this form, the partition function appears as the expected gradient over samples from the model. Although the expectation of this gradient is just as difficult to compute as the partition function itself, this form suggests a stochastic optimization approach: we can estimate these expectations from as many or as few samples as we like, so long as they come from the right distributions.

Contrastive Divergence (CD) algorithms use a *Markov Chain* to draw samples from the model to estimate the expected gradient under the model distribution, while simultaneously drawing samples from the training set to estimate the expected gradient under the natural distribution (Hinton, 2002). The Markov chain typically used to train RBMs is based on a *Gibbs sampling* scheme. Gibbs sampling is a technique for drawing samples from any probability model: take turns updating each variable by sampling from its conditional probability given the current value of all the other variables. In the RBM we consider that there are two vector-valued variables: x and h. We can repeatedly sample $h \sim P(h|x^{(t)})$, and then $x^{(t+1)}) \sim P(x|h)$. Sampling $x^{(t)}, x^{(t+1)}, x^{(t+2)}, ...$ in this way yields (eventually, at equilibrium) points that are distributed according to $P_\theta(x)$.

Specific variants of CD differ in exactly how they handle the Gibbs sampling Markov chain. Often what is called Contrastive Divergence refers more specifically to the CD1 algorithm, in which Markov chains are started from training examples and run for only a single step. Similarly the CD-K algorithm runs Markov chains for K steps starting from training examples. The Persistent CD (PCD) algorithm, which is also known as Stochastic Maximum Likelihood (SML) learning, runs the Markov chains properly, without restarting them at all (Tieleman, 2008). In PCD the samples used to estimate the expected gradient with respect to the model distribution tend to be more correlated from one time step to the next (when they are correlated we say the model does not *mix well*) but in the long run, PCD gives an unbiased estimate of the gradient whereas CD1 and CD-K give biased estimates. The PCD and CD-1 algorithms are used to train RBM models in Chapter 5.

2.5.12 Gaussian RBM

The Gaussian RBM (GRBM) is like an RBM, but it is defined for $x \in \mathbb{R}^N$ rather than $x \in \mathbb{B}^N$ (Bengio et al., 2007; Hinton and Salakhutdinov, 2006; Krizhevsky, 2009; Welling et al., 2005). It is parametrized by a *weight matrix* $V \in \mathbb{R}^{N \times K}$ two *bias vectors* $a \in \mathbb{R}^N$ and $b \in \mathbb{R}^K$, and a *precision matrix* $\Lambda \in \mathbb{R}^{N \times N}$.

$$E(x,h) = 0.5x'\Lambda x - a'x - h'Vx - b'h \tag{2.46}$$

Typically the precision matrix Λ is chosen to be a diagonal one because it must be inverted for Gibbs sampling. It is trained just like the basic RBM, but the conditional distribution of $P(x|h)$ is different on account of the $0.5x'\Lambda x$ term. The Gaussian RBM forms the basis of the Spike and Slab RBM presented in Chapter 5.

2.5.13 Convolutional Architectures

The discussion thus far about learning features has stayed very general. Images have been just vectors in some Euclidean space, and densities over that space have been Gaussian or Gaussian Mixtures. But we have considerable prior knowledge about images, and convolutional architectures are now a popular way of combining that prior knowledge with this general mathematical perspective (LeCun et al., 1989, 1998a).

For our purposes, a 2D convolution is a binary operator whose operands are an image $u \in \mathbb{R}^{R \times C}$ and a *filter*, $v \in \mathbb{R}^{S \times T}$.[2] The result $y \in \mathbb{R}^{(R-S+1)\times(C-T+1)}$ of what is called a *valid-mode* convolution is written as $y = u * v$ and defined as

$$y_{i,j} = \sum_{k=1}^{S} \sum_{l=1}^{T} u_{(i+k-1),(j+l-1)} v_{(S-k+1),(T-l+1)}. \tag{2.47}$$

Each result $y_{i,j}$ is the result of multiplying the image u at position i, j by an inverted version of a smaller filter image v.

2. The "filter" is sometimes called a "kernel" in discussion of convolution, but since it has no connection to kernel-based features, I will use the term "filter".

A convolutional neural network is a neural network with a matrix multiplication replaced by a convolution. For example, taking a basic neural network (Equation 2.29) as a point of departure, something like

$$l^* = \arg\max_{l \leq L} \left(W \tanh(x * v + c) + b\right)_l , \tag{2.48}$$

would be a single-layer convolutional network, except that our notation needs some refinement. In Equation 2.29 x was a vector, and now that $x * v$ is a matrix what does it mean to add a vector c of biases? Recall that in preprocessing images for learning (Section 2.4.1), the last step was to rasterize an image from its original matrix representation to a vector. To use convolutional architectures we must omit that step, and leave our images in their more natural matrix representation. For convolutional architectures we must likewise un-rasterize the bias vector c so that it too reflects the 2D structure of the convolution output. Ultimately the linear classifier (W, b) requires a vector (not a matrix) of features, so in convolutional architectures we still have to perform a rasterization step at some point. Often this rasterization is done just prior to classification by W, but in deep convolutional networks, the rasterization can be done just as well at any intermediate layer.

2.5.14 Sparse Coding

Sparse coding represents an alternative to the RBM and Parzen windows for learning image representations that can be used as features (Gregor and LeCun, 2010; Kavukcuoglu et al., 2009; Lee et al., 2008b; Ranzato et al., 2008). The objective in sparse coding is to explain (i.e., code) a signal (i.e., an image) $x \in \mathbb{R}^N$ as the linear combination of a small number of *dictionary elements* among the columns of $B \in \mathbb{R}^{N \times K}$. There are two important kinds of sparse coding, which correspond to different formalizations of the notion of "a small number" of elements in $z \in \mathbb{R}^K$. They can be seen as two different regularizations on a simple linear generative

model of the data:

$$\mathscr{L}^{(\text{l1})}(x;B) = \min_{z \in \mathbb{R}^K} ||x - Bz||_2^2 + \alpha \sum_i |z_i| \qquad \ell_1 \text{ sparse coding} \tag{2.49}$$

$$\mathscr{L}^{(\text{Cauchy})}(x;B) = \min_{z \in \mathbb{R}^K} ||x - Bz||_2^2 + \alpha \sum_i \log(1 + \gamma z_i^2) \qquad \text{Student-t sparse coding} \tag{2.50}$$

If we interpret these loss functions as being of the form $-\log P(x|z)P(z)$, the first formulation corresponds to choosing a Laplacian prior over the elements z_i, whereas the second formulation corresponds to a Student-t distribution over elements $\sqrt{\gamma} z_i$. The columns of B are typically constrained to have unit ℓ_2-norm to ensure the sparsity penalty on z is meaningful. Learning by sparse coding means adjusting (by gradient minimization methods) the matrix B of dictionary elements to minimize expected loss, be it $\mathscr{L}^{(\text{l1})}$ or $\mathscr{L}^{(\text{Cauchy})}$.

Sparse coding has been advanced as a principle for why V1 simple cells behave as they do (Cadieu, 2009; Cadieu and Olshausen, 2009; Karklin and Lewicki, 2008; Olshausen and Field, 1996, 1997). The dictionary elements B_i learnt by sparse coding of whitened image patches resemble small edges like the ones that tend to activate simple cells. The sparse coding theory is attractive because the creation and propagation of neural spikes takes physical energy, and it stands to reason by the laws of thermodynamics that all else being equal, the brain should use no more spikes than necessary. At the same time, if spikes transmit information then that information must be preserved. The fact that sparse coding yields V1-like receptive fields should not be taken as very strong evidence that sparse coding *per se* is the learning principle in the brain, because a number of other latent variable models discussed here such as various flavours of RBM, as well as latent-variable approaches not in this chapter such as auto-encoders, K-nearest neighbours, independent components analysis, all yield similar Gabor-like edge filters. Some models even operate in an arguably sparse regime, even though sparsity was not a training

objective. There have also been challenges to the idea that cortex optimizes sparsity either in learning or in inference of the codes for stimuli (Berkes et al., 2009b). Despite these caveats, sparse coding continues to be a successful and influential model for the organization of the visual system.

2.5.15 Slow Feature Analysis

Slow Feature Analysis (SFA) is an algorithm for modelling not images, but pairs of consecutive images ($u \in \mathbb{R}^N,\ v \in \mathbb{R}^N$) from movies. The objective in SFA is to identify a linear basis $B \in \mathbb{R}^{N \times K}$ for images that leads to a stable representation $y \in \mathbb{R}^K$, $z \in \mathbb{R}^K$ over time. SFA is the solution of the following minimization problem:

$$\mathscr{L}^{(\mathrm{SFA})}(u, v; B) = \min_{y \in \mathbb{R}^K,\ z \in \mathbb{R}^K} ||u - By||_2^2 + ||v - Bz||_2^2 + ||y - z||_2^2, \tag{2.51}$$

$$B^{(*)} = \underset{B \in \mathbb{R}^{K \times N} \text{ s.t. } ||B_i||_2 = 1\ \forall i}{\arg\min} \mathbb{E}_{(u,v) \sim \mathscr{G}_{u,v}} \left[\mathscr{L}^{(\mathrm{SFA})}(u, v; B) \right]. \tag{2.52}$$

There are fast algorithms for performing SFA that find $B^{(*)}$ by solving a generalized eigenvalue problem (Berkes and Wiskott, 2005). Slow feature analysis has been advanced as a principle to explain why V1 complex cells behave as they do (Berkes and Wiskott, 2005; Cadieu and Olshausen, 2009; Körding et al., 2004), and even as an explanation for the formation of place cells in the hippocampus that are believed to play a role in localization and navigation (Franzius et al., 2007; Wyss et al., 2006). The idea of regularizing representations to be stable over time is appealing, and motivates the learning algorithm presented in Chapter 4.

2.5.16 Independent Components Analysis

Independent Components Analysis (ICA) is a principle for factor analysis, which seeks to represent observed vectors x as a linear combination B of coefficients z ($x = Bz$) so that the marginal distributions of elements $z_i|x$ are as independent as possible. This is possible because of the law of large numbers, which implies that a linear combination of independent non-Gaussian *source* variables is more

Gaussian than any of the sources. Different ways of measuring non-Gaussian-ness lead to different ICA objectives, but they all seek a basis $\boldsymbol{B}$ that maximizes the non-Gaussian-ness of the marginal distributions of z_i (Bell and Sejnowski, 1995; Hoyer and Hyvärinen, 2000; Hyvärinen and Hoyer, 2001; Hyvärinen et al., 2001; Karklin and Lewicki, 2003). Hyvärinen (2009) reviews how ICA models can account for various aspects of simple cells, complex cells, and the retinotopic organization of V1.

This concludes the review of visual system anatomy, physiology, computer vision, and machine learning. The following chapters describe my own contributions in machine learning.

CHAPTER 3

COMPLEX CELLS FOR CLASSIFICATION

Title	Suitability of V1 Energy Models for Object Classification
Authors	James Bergstra, Yoshua Bengio, and Jérôme Louradour
Publication	Neural Computation, Vol. 23, No. 3: 774–790, March 2011.

Simulations of cortical computation have often focused on networks built from simplified neuron models similar to rate models hypothesized for V1 simple cells. However, physiological research has revealed that even V1 simple cells have surprising complexity. Our computational simulations explore the effect of this complexity on the visual system's ability to solve simple tasks, such as the categorization of shapes and digits, after learning from a limited number of examples. We use recently proposed high-throughput methodology to explore what axes of modeling complexity are useful in these categorization tasks. We find that complex cell rate models learn to categorize objects better than simple cell models, without incurring extra computational expense. We find that the squaring of linear filter responses leads to better performance. We find several other components of physiologically-derived models do not yield better performance.

3.1 Introduction

An important role of the visual system is to transform retinal signals so that object categorization can be carried out in higher cortical areas such as V4 and IT (Dayan and Abbott, 2001; Serre et al., 2007a). However it remains unclear, even in V1, what transformations individual neurons perform (Doi and Lewicki, 2007; Haüsser and Mel, 2003; Olshausen and Field, 2005)

Many rate models have been proposed for V1 based on studies of spike-triggered averaging (Hubel and Wiesel, 1962; Nykamp and Ringach, 2002). The simplest rate model is a linear filter whose output has been rectified to be non-negative and bounded. Rectification has been carried out using the *logistic sigmoid* (Kouh and Poggio, 2008; Nykamp and Ringach, 2002; Rumelhart et al., 1986b; Wilson and Cowan, 1972) or the Naka-Rushton equation (Heeger, 1992; Naka and Rushton, 1966) to describe V1 *simple cells* (first described in Hubel and Wiesel, 1962). Other cells (called *complex cells*) in V1 have been found to respond robustly to narrow bars of light in nearby positions, but not to their superposition. This behavior cannot be explained by a linear model. The classic model for complex cells is the *energy model*, in which a complex cell response is modeled by a sum of squared responses from a number of afferent simple cells (Adelson and Bergen, 1985; Dayan and Abbott, 2001). More recently, the distinction between simple and complex cells has been challenged by findings that cells in V1 span a more continuous range of behavior that also includes max-like integration by complex cells of their afferent inputs (Finn and Ferster, 2007; Kouh and Poggio, 2008; Riesenhuber and Poggio, 1999; Rust et al., 2005; Serre et al., 2007a).

But what is all this modeling capacity *for*? In contrast to research that is based on [cross-]correlation analysis, we advance a different criterion for comparing models of the visual system. Part of the visual system can be interpreted as implementing a function from images to object categories. This function adapts over time as the brain learns to categorize particular objects, and to generalize about categories. The rules that govern this adaptation over time comprise a *learning algorithm* which

we would ultimately like to understand better. A fundamental result in learning theory is that learning algorithms have preferences, also known as inductive biases, priors over functions (Vapnik, 1995). This is true even of learning algorithms with the capacity to approximate any continuous function. Consequently, different learning algorithms produce different functions, even when presented with the same data. In the case of the brain's learning algorithm for vision, the priors of that algorithm play a central role in our ability to learn the structure and invariances in the images that we see. Whenever we choose a model of the visual system and a procedure for setting the internal parameters of that model, we also implicitly choose a learning algorithm and induce a particular prior over functions. Using the approach of rational analysis, we suppose that the brain's visual system is optimal at learning from limited data, and we can understand the learning algorithm for vision by studying the functional priors that support rapidly learning to categorize objects (Anderson, 1990). If that model and fitting procedure is faithful to the visual system, then the prior of that model will match the prior of the visual system, and consequently the model will learn with the same competences and weaknesses as the visual system.

In this work, we compare many mathematical models of V1-like neurons as bases for object categorization. Since we lack a complete characterization of the functional prior of the visual system's learning algorithm, we chose categorization tasks that seem trivial from the perspective of an adult human: distinguishing simple geometric shapes, digits 0-9, and five kinds of toy. Learning theory gives us the terminology to be more precise about what *trivial* means here. It means that the learning algorithm of the human visual system has a prior preference for functions that work well for these tasks. In the style of Cox et al. (2008), we mix and match different parametric ingredients that have been put forward to explain simple and complex cells within the energy model framework, to see which of those ingredients is important for generalizing about objects. We find that:

– complex-like elements (involving sums of squared linear filter responses) induced better priors for object classification than simple-cell elements,

- a polynomial saturating non-linearity (based on division) was generally better than an exponential non-linearity (i.e. tanh, logistic sigmoid) which is common in the machine-learning community,
- numerically optimizing the exponent used to pool the multiple filters of a complex-like cell was not useful - squaring (2) was the best choice,
- the possibility of reweighting numerator terms as suggested in Kouh and Poggio (2008) was slightly helpful in one task and slightly harmful in another,
- multi-filter models based on complex-like V1 elements required fewer hidden units; so although each complex-like element was more expensive to compute, the total cost of training the best complex-like models was similar to the cost of training the best simpler V1-like models.

This work differs from that of Shams and von der Malsburg (2002) and Cox et al. (2008) in that we used machine learning methods to tune our V1-like elements, and we classified images rather than decoding their internal representations. The design of our experiments is similar to that of Edelman et al. (1997); our models and tasks are different, but our finding that complex-like behaviour in V1 is important for successful learning agrees with their findings.

3.2 High-Throughput Screening of V1-like Models

For our study we defined a single broadly-encompassing parametrization of a V1-like model cell response (Equation 3.1) and instantiated particular V1-like models by restricting the general form in various ways. Equation 3.1 is an extension of the Canonical Neural Circuit of Kouh and Poggio (2008).

$$R = \sigma\left(\frac{\sum_k^K (-1)^k (\sum_j^J w_{jk} a_{jk}(x)^{p_k})^{r_k}}{\beta + \gamma \sum_k^K (\sum_j^J a_{jk}(x)^{q_k})^{s_k}}\right). \tag{3.1}$$

This general form encompasses the various models presented in Section 3.1 (see Table 3.I) by various choices for scalars $(J, w_{jk}, p_k, q_k, r_k, s_k, \beta)$, compressive non-linearity σ, and activation function $a_{jk}(x)$ of the stimulus (input). We allow for a_{jk} to be either an affine function $a_{jk}(x) = v_{jk}x + b_{jk}$ or a half-rectified affine

function $a_{jk}(x) = \max(v_{jk}x + b_{jk}, 0)$, in which v_{jk} is a *weight vector* or filter, and b_{jk} a corresponding scalar-valued bias. The x here is a rasterized image vector of pixel values. The following few paragraphs illustrate how this canonical form can be instantiated to immitate or match other models.

We can get feed-forward simple-cell-like models a few ways. For example, with $K = J = 1$ and all p, and q r, s, w all 1 as well we have a single-filter model cell without any squaring or pooling. Choosing $a_{jk}(x) = v_{jk}x + b_{jk}$, $\sigma(z) = \tanh(z)$, $\beta = 1$ and $\gamma = 0$, yields $R = \tanh(v_{jk}x + b_{jk})$; this is the logistic sigmoid unit popular in neural network classification. We can get another sigmoidal activation function by choosing $\sigma(z) = z$ as well as $\beta = 1$ and $\gamma = 1$ (Kouh and Poggio, 2008).

The energy model of Adelson and Bergen (1985) coincides with the setting of $K = 1$, $J = 2$, $p = q = 2$, $w = 1$, and choosing $a_{jk}(x) = v_{jk}x + b_{jk}$. Various combinations of non-linearity and settings of r and s (e.g. 0.5 and 1) are consistent with the mechanism and idea of pooling across a quadrature pair.

The parametrization reduces to the Canonical Neural Circuit (Equation 2.1 in Kouh and Poggio (2008)) when $K = 1$, all a_{jk} are affine, σ is the identity and when γ is 1. The canonical circuit can immitate max-pooling with a large value for p_k paired with $r_k = p_k^{-1}$, and our parametrization can do so in the same way.

We include the possibility that $K = 2$, and the possibility of half-rectification in functions $a_{jk}(x)$ to accommodate the model of Rust et al. (2005). Their model coincides with $K = 2$, $J \geq 3$, $p = q = 2$, $r = s = 0.5$, $\sigma(z) = z$. In their model, one function a_{jk} is half-rectified among the set for which $k = 1$, and none of the activations a_{jk} is half-rectified among those for which $k = 2$.

3.2.1 Adapting Model Parameters

We evaluated V1-like models by forming a population of N V1-like elements (hidden units) and feeding their outputs into a logistic classifier to form a one-hidden-layer neural network. Each hidden unit R_n was fully connected to a small grey-scale image stimulus with initially random, but eventually learnt weights (the v_{jk} and b_{jk} for each V1-like element). The model categorized objects by activating

categorization neurons, each of which was a linear function of the full set of V1 model neurons. There was one categorization neuron for each object category. The tasks were object discrimination tasks, and they are described in Section 3.3. To minimize Ω in Equation 3.3, training pushes the real-valued confidence y_{z_t} to be the largest of all confidences y_l. At test time, the category with the largest confidence y_l was deemed to be the predicted category.

$$y_l = c_l + \sum_{n=1}^{N} A_{l,n} R_n(x_t) \qquad \text{confidence that } x_t \text{ has label } l \tag{3.2}$$

$$\Omega_t = -y_{z_t} + \log\left(\sum_{l=1}^{L} e^{y_l}\right) \quad \text{error (neg. log. likelihood) predicting label } z_t \text{ for } x_t$$

$$\Omega = \sum_{(x_t,z_t)\in\{\text{Train}\}} \Omega_t \qquad \text{error on training data} \tag{3.3}$$

For a given V1-like model we searched for the best filters and other model parameters (e.g. activation function weights v_{jk} and b_{jk}, categorization weights $A_{l,n}$) by stochastic gradient descent (Rumelhart et al., 1986a). We denote the V1-like population response to a stimulus $x_t \in \mathbb{R}^M$ by a vector R whose N elements are called R_n, for $n \in [1,N]$. Each of L categorization neurons y_l is affine in R, with weight matrix $A_{l,n}$ and bias c_l, as in Eq. 3.2. Integer z_t denotes the correct category of x_t. We initialized A, c to zero and the internal parameters of model neurons R to random values, and then iteratively adjusted them to minimize the error (Ω) on small samples of training data. In trials, where w (in eq. 3.1) was not fixed to 1, the numerator weights w were also optimized by gradient descent. In some trials the scalars p_k, r_k, s_k, q_k were optimized by gradient descent too. Filter values were initialized uniformly within a range $(-\sqrt{\frac{6}{N+M}}, \sqrt{\frac{6}{N+M}})$ about zero, as recommended in Bengio and Glorot (2010). The learning rate (the proportion of the negated error gradient by which parameters were incremented on each iteration)

was sampled alongside the rest of the trial hyper-parameters. The seed used to initialize the filter in each a_{jk} transformation was sampled randomly for each trial.

We compared models by dividing the data from each task into three sets: training, validation and test (Bishop, 1995). The training data were used to calculate the fitting criterion Ω and its gradient with respect to model parameters. The validation data and test data were used to estimate the out-of-sample classification performance of the fitted system with each number of V1-like neurons (by counting what fraction of objects were categorized correctly). This validation set score was used in an early stopping heuristic to decide how much gradient descent on the training set was enough, and was also used to select among models in the empirical results described in Section 3.4. The early stopping heuristic was to wait at least 20 iterations through the training data, and then stop when twice as much training had been done, as had been necessary to arrive at the best model up to that point. The model scores listed in Section 3.4 are all test set scores.

The models classes presented here are defined in terms of their functional form, without reference to the kind of filters that determine the response in eq. 3.1. In analysis of V1 recordings, these filters are typically Gabor-like, with localized receptive fields and pairs of squared filters that implement quadrature pairs (Dayan and Abbott, 2001). Our experiments explore why this kind of filter arises, so we do not initialize our filters with Gabor-like patterns. Our experiments involve tuning randomly initialized filters by supervised learning of tasks that V1 neurons are able to perform, in order to compare models of what V1 neurons do.

3.2.2 Hyper-parameter Sampling Distribution

The set of trials (V1-like models and hyper-parameters) encompassed by our parametrization is too large to search with a grid, so we adopted the high-throughput methodology of Cox et al. (2008); Pinto et al. (2009) to explore the hyper-parameter space. High-throughput search requires a proposal distribution from which to sample models. Essentially, we draw models from a distribution rather than from locations on a grid. One advantage of random draws is that if we project onto any

one axis of the hyper-parameter space (e.g. the learning rate) then the random draws will cover the legal range more uniformly, whereas a grid will test just a few values. Another advantage of random draws is that if there is independence between the effect of two hyper-parameters, then random sampling explores both simultaneously – this is a much more efficient search. We postulate that several hyper-parameters in Equation 3.1, such as the choice of σ, the half-rectification of each a_{jk}, and some of the choices about learning with this model have almost independent effects. To the extent that hyper-parameters are independent, a high-throughput random search is more statistically efficient. We as experimenters do not have to specify or know exactly which hyper-parameters are independent.

For our high-throughput search, we sampled hyper-parameter assignments according to the distribution in Figure 3.1. Some combinations of hyper-parameters lead to unusable models. For example, when a_{jk} are not rectified, they may be negative, and a non-integer exponent will lead to a complex-valued response. We rejected such hyper-parameter assignments when they were sampled. There is also an over-parametrization if $J > 1$ and either $p_k = 1$ or $q_k = 1$; in this case the model is equivalent to a model where $J = 1$ so we rejected these hyper-parameter assignments as well. We also rejected assignments in which p_k and r_k (similarly q_k and s_k) were both greater than one. Such assignments do not correspond to a V1-like model in the literature, and they are not numerically stable during learning. We also rejected assignments in which squashing was done by division ($\gamma = 1$ and $\sigma =$ identity) but the denominator could approach or equal zero because these models were also numerically unstable. Some model-family frequencies under the [post-rejection] sampling distribution are given in Table 3.I.

3.3 Discrimination tasks

We measured the ability of each visual system model to learn three object categorization tasks: **Shapes**: triangles, squares, or circles; **Digits**: 0,1,2,...9; or five kinds of small **Toys**. **Shapes** images (32×32 pixels) were generated by

- How many outer terms? $K \sim U(1,1,1,1,2)$
- How many inner terms? $J \sim U(1,1,2,2,3,5)$
- Squashing: $\sim U(\sigma = \text{tanh and } \gamma = 0[\frac{1}{3}], \sigma = \text{identity and } \gamma = 1[\frac{2}{3}])$
- Exponents fixed (Fix) or optimized (Opt)? $\sim U(\text{Fix } [\frac{3}{4}], \text{Opt } [\frac{1}{4}])$
- Filter exponent $p_k \sim U(1,1,2,2,3)$ for each k
- Filter exponent $q_k \sim U(1,1,2,2,3)$ for each k
- Norm exponent $r_k \sim U(1,2,p_k^{-1})$ for each k
- Norm exponent $s_k \sim U(1,2,q_k^{-1})$ for each k
- Number of activation functions to half-rectify $\sim U(0,1,J)$
- Numerator weights: vectors w_{*k} are all 1 $[\frac{1}{2}]$ or are non-negative and sum to 1 $[\frac{1}{2}]$
- Number of hidden units $N \sim \frac{2^{\mathscr{U}(4,12)}}{JK}$ i.e. 16 - 2048
- Learning rate: $2^{-\mathscr{U}(4,9)}$
- L_1 filter regularization 0 $[\frac{3}{4}]$ or else $2^{-\mathscr{U}(4,10)}$ $[\frac{1}{4}]$
- L_2 filter regularization 0 $[\frac{3}{4}]$ or else $2^{-\mathscr{U}(4,10)}$ $[\frac{1}{4}]$
- The scalar β was fixed at 1.

Figure 3.1: Sampling distribution over V1-like models and learning algorithm hyper-parameters for high-throughput search. Random variables (such as the Learning rate) that were chosen uniformly in a range from a to b are denoted $\mathscr{U}(a,b)$. Random variables chosen from among a few (potentially repeating) values (a,b,c) where each value is equally likely are denoted $U(a,b,c)$. Random variables chosen from among a few values where each value is not equally likely are denoted $U(a[P(a)],b[P(b)],c[P(c)])$. The distributions for these variables are shown as being independent, but a rejection policy introduced dependence between them. For example, we rejected models that might raise a negative number to a fractional power, and models that might divide by a zero denominator.

Table 3.I: Sampling probabilities of various V1 models under our hyper-parameter distribution. The top part of the table shows what percentage of our randomly sampled models correspond to selected models from the literature. HPU (Higher-order Processing Unit) is a tanh of a polynomial of x (Rumelhart et al., 1986b). The standard sigmoid model is a linear filter squashed by the tanh function. Sigmoid-like models have a possibly-rectified linear filter squashed by either tanh or division ($\gamma = 1$). The Canonical Neural Circuit (Kouh and Poggio, 2008) includes max-pooling and energy models by a choice of exponent values within the model. The bottom part of the table lists the percentage of randomly sampled models that come out with various properties. Our distribution is designed to compare many of the elements introduced in Kouh and Poggio (2008) with simpler models ($JK = 1$), but we also include hyper-parameter K so that $K = 2$ permits the subtractive and divisive inhibition suggested in Rust et al. (2005).

Model	Frequency
Kouh and Poggio (2008)	56.4%
sigmoid-like	13.0%
HPU	5.1%
Rust et al. (2005) (no divisive inhibition)	4.5%
Adelson and Bergen (1985)	2.5%
standard-sigmoid	1.1%
Rust et al. (2005) (with divisive inhibition)	0.2%
$JK = 1$	42.0%
$p_k = 2 \quad \forall k$	35%
$K = 2$	11.6%

varying the type of shape, the position, size, orientation, and grey scales of the foreground and background [1]. **Digits** (32×32 pixels) was the MNIST database of hand-written digits [2]. **Toys** (32×32 pixels) was a modified "small NORB" dataset [3]. The five sorts of toys that the visual system models had to distinguish were four-legged animals, human figures, airplanes, trucks, and cars. We modified the public dataset for our experiments by shuffling all the toy instances together, and drawing 5000 training examples, 14440 validation examples, and 29160 testing examples randomly without replacement. The roles of these training, validation, and testing examples is explained in Section 3.2.1. Sample stimuli for each of these tasks are illustrated in Figure 3.2.

3.4 High-Throughput Evaluation

We sampled 1000 models for each of our three datasets and analyzed how the performance in our tasks correlated with modeling choices.

The best models found in the random search produced scores competitive with the state of the art. The best **Digits** model in the random search scored 1.56% error, the best score on **Toys** was 1.78%, and the best score on **Shapes** was 3.1% error.

The effect of each modeling choice on performance is illustrated in Figure 3.3. Each panel in Figure 3.3 shows the five best models (in validation) for each restriction in a single model hyper-parameter. Trials are characterized by their classification error rates on test data. There were 5000 test examples in **Shapes**, and the best models had around 4% error, so bear in mind while reading the following section that *relative* differences in performance of more than 16% are statistically significant at a 95% confidence level. There were 29160 test examples in **Toys** and scores were around 2% error so relative differences in performance of 10% are

1. Data available at `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/BabyAIDatasets`
2. Data available at `http://yann.lecun.com/exdb/mnist/` (LeCun et al., 1998a)
3. The original data are available at `http://www.cs.nyu.edu/~ylclab/data/ norb-v1.0-small` (LeCun et al., 2004)

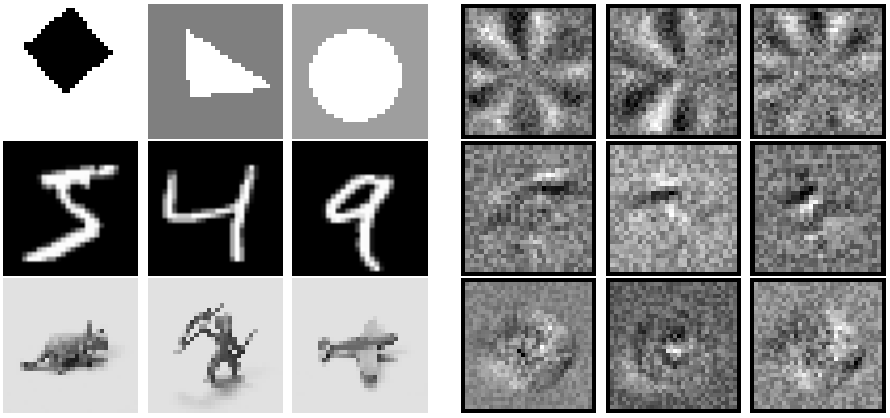

Figure 3.2: *Left:* Images from each of the three datasets: **Shapes**, **Digits**, **Toys**. *Right:* Filters v_{jk} learnt by the best models in our study on the respective datasets. These filters come from models with multiple filters per neuron, and with squared filter responses. Filters were chosen to be representative of the population in the learnt model. For the shapes dataset, the model has learnt to phase-offset filters that implement angle-invariant selectivity for edges radiating out from near the middle of the image. For the digits, the model has learnt oriented Gabor-like edge detectors; here the squaring of filter responses makes model neurons invariant to edge polarity. For the toys, the model has learnt filters with circular swirling patterns. The mechanism of these swirling filters is not clear, but they serve to generalize well to the validation and test data.

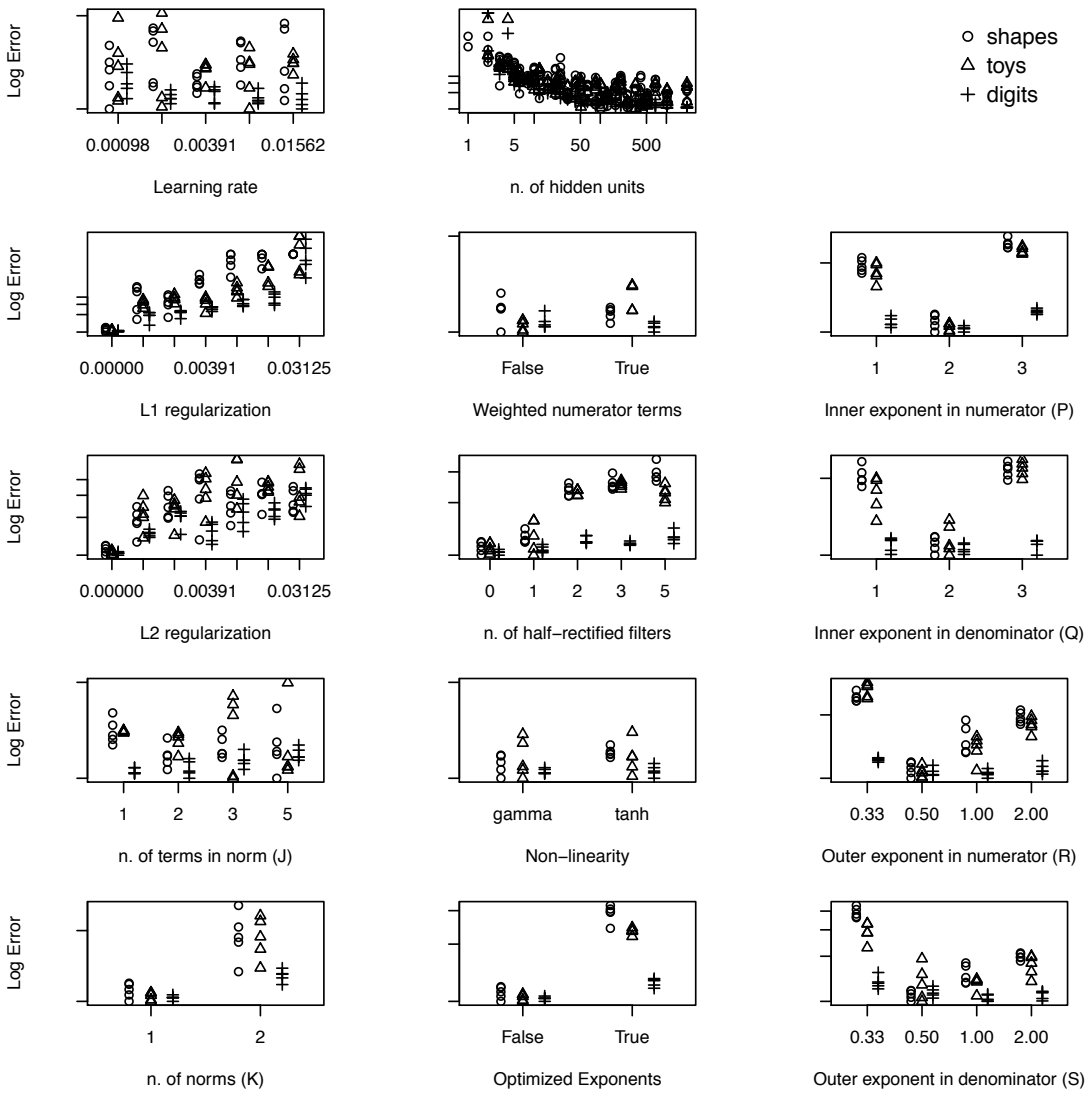

Figure 3.3: Each panel shows test set scores of the 5 best (in validation) models in each category on each dataset relative to the best model on each dataset, from a broad random search over 1000 general V1 models (Eq. 3.1). The y-values are log-scaled, and each dataset's scores are shifted vertically so that they are aligned at the bottom of each panel. The y-axis ticks correspond to a doubling of test-set error. Panels for p, q, r and s show performance as a function of the maximum (initial) value of p_k, q_k, r_k and s_k respectively. The standard error in the best test-set error rates plotted on the vertical axes are at most 10%, which is at most approximately the size of the icons.

significant at the 95% level. There were 10000 test examples in **Digits** and scores were around 1.5% error so relative differences in performance of 19% are significant at the 95% level. At an 80% confidence level, differences of 8% on **Shapes**, 5% on **Toys** and 10% on **Digits** are significant.

Regarding the number of norms (K), we found that the top five models for all three datasets when $K = 1$ were better than the top five with $K = 2$. The best score with $K = 2$ were 34% worse for **Shapes**, 39% worse for **Toys** and 18% worse for **Digits**. Part of the reason for better performance from $K = 1$ models is that there were more of them–90% of trials were with $K = 1$. Still, the additional norm in the numerator and denominator ($K = 2$) conferred no clear advantage.

Regarding the number of terms in each norm (J), the best results on **Digits** were with $J = 2$, **Toys** were with $J = 3$ and **Shapes** were with $J = 5$. For both **Toys** and **Shapes** the single-filter model ($J = 1$) was significantly poorer (by 27% on **Shapes**, by 40% on **Toys**) than the best model. There were approximately equal numbers of trials with $J = 1$ as with $J > 1$.

With regards to filter exponents p_k and q_k, we found that squared filters ($p_k = 2$ and $q_k = 2$, $\forall k$) gave the best performance. Linear filters ($p_k = q_k = 1 \forall k$) were at best 60% worse than squared filters on **Shapes** and **Toys**, and cubed filters ($p_k = q_k = 3 \forall k$) were at best 95% worse on **Shapes** and **Toys**. On the **Digits** task, linear filters for p_k were 8% worse and cubic ones were 20% worse. On **Digits** there was no difference among values for q_k. Trials were split such that roughly 40% had $p_k = 1$, 40% had $p_k = 2$, and 20% had $p_k = 3$. The distribution of q_k trials was similar and independent from p_k.

Regarding norm exponents r_k and s_k we found that $\frac{1}{2}$ (a square root) was the best on **Shapes** and **Toys**. Values of $\frac{1}{3}$ and 2 gave performances that were at best 60% worse. A value of 1 was 30% worse on **Shapes** but just 8% worse on **Toys**. On **Digits** $s_k = 1$ and $s_k = 2$ were best by 10% over $\frac{1}{2}$ and all values except $\frac{1}{3}$ were equally good for r_k.

To test the potential advantage of fractional exponent values close to $p_k = q_k = 2$ and $r_k = s_k = .5$ we looked at trials where the exponent was adjusted according to

the gradient during learning. The panel labelled "Optimized Exponents" illustrates that performance with these learnt exponents was always worse (140% worse on **Shapes**, 120% worse on **Toys**, and 20% worse on **Digits**), and comparable to the performance when p or r were fixed to non-optimal values.

Half-rectification of filters (a_{jk}) was harmful. Approximately half the trials had 1 half-rectified filter, and the other half were divided about evenly between $0, 2, 3$, and 5. The best trials had 0 half-rectified filters, and trials with 2 or more half-rectified filters were 120% worse on **Shapes**, 120% worse on **Toys** and 14% worse on **Digits**.

In approximately 30% of trials (50% of the trials where $J > 1$), we optimized the weighting of numerator terms (vectors $w_{\cdot k}$) by gradient descent, under the constraint that they be non-negative and sum to 1. Optimization of these vectors helped to reduce error in the **Shapes** dataset (by a factor of approximately 12%) but it raised the error rate in **Toys** (by about 17%) and did not make any difference in **Digits**.

Approximately two thirds of trials used squashing by division ($\gamma = 1$) rather than tanh. The best-performing of tanh trial on **Shapes** was 20% worse than the best division model, but on the other two datasets the difference in performance was not significant. There was less variability among results when squashing by division.

The choice of learning rate was not critical good results were obtained for each task with every value in the range under investigation. ℓ_1 and ℓ_2 regularization of weights was simply harmful.

3.4.1 Single-filter Models vs. Multi-filter Models

One of the basic questions our experiment addresses is whether modelling capacity is better spent on additional model neurons, or on more complicated multi-filter model neurons. One way to quantify the capacity of a model is by the number of degrees of freedom that may be adapted. In our case, this quantity is dominated by the product of three terms: the number of hidden units (N, see Eq. 3.1), the num-

ber of filters in each norm (J), and the number of norms (K). Figure 3.4 illustrates the relationship between capacity and performance in these models, for single-filter models ($JK = 1$) and multi-filter alternatives ($JK > 1$). Two things stand out in Figure 3.4: a certain amount of capacity is necessary for good performance (there are no points in the bottom left quadrant), and it is important that the capacity be in the form of model neurons with multiple linear filters (solid icons dominate the bottom right quadrant). So ultimately, additional single-filter model neurons are a poor substitute for multi-filter alternatives when filters must be learnt from data.

Figure 3.4 was computed as follows. For each dataset, we drew a random sample of 1000 model trials. Each random sample of 1000 model trials was partitioned into *condition-sets* according to the number of filters in the model (product JKN), rounded to the nearest power of 2. The figure shows the results of each dataset slightly offset horizontally for clarity. Each condition is a pair (dataset, capacity) and each condition-set is the set of trials matching the condition, but with different non-linearities, learning rates, exponents, and so on (see variations in Figure 3.1). We define a test-set score for each condition by taking the test-set score from the best model (according to validation performance) in its condition-set. The smallest condition set had 39 elements, and condition sets with more than 50 elements were randomly truncated to 50. Each test score was divided by the best test score in the original 1000 trials so that performance could be compared between datasets. Conditions whose test-set error was high (more than twice the best test score on the same dataset) are also not shown for clarity.

For all the models in our study, the computational cost of determining the label for a test example is proportional to the number of filters, which is the horizontal axis (capacity) of Figure 3.4. The best-performing multi-filter models required approximately the same amount of computation time to train and test as the best single-filter models that were a few times larger, but the multi-filter models performed better.

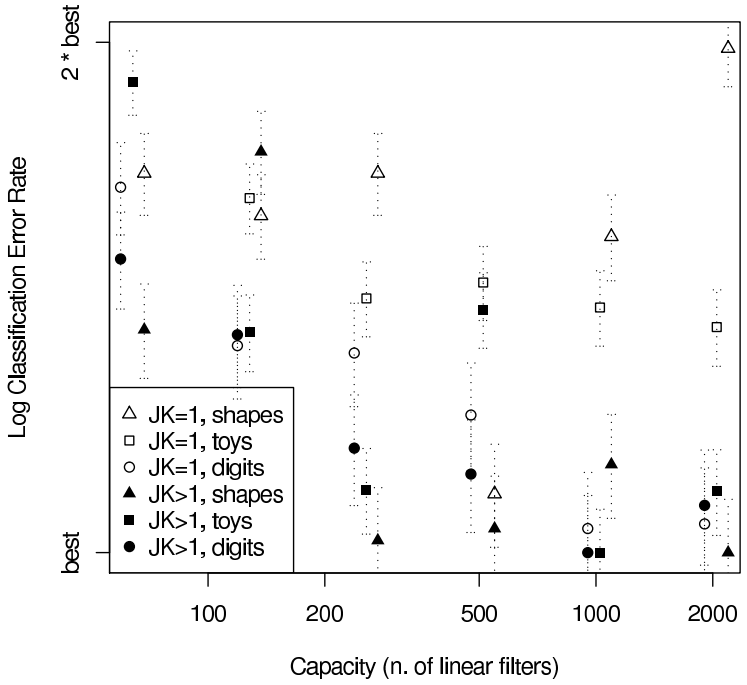

Figure 3.4: Why not just use more simple model neurons with one filter each? There are different ways to add capacity to a model. Each point below corresponds to test error for the best model in one experimental condition (dataset and capacity, defined as $N \times J \times K$ see Eq. 3.1). The horizontal axis is capacity (with each dataset's results slightly offset for clarity), and the vertical axis is out-of-sample classification error relative to the best model on the same dataset. Each best model is chosen over N (with $37 < N < 50$) other randomly sampled models within a given condition (different learning rate, different exponents, see Fig. 3.1 for all variations). Solid markers denote the performance of models with $JK > 1$, outlined markers denote the performance of models with $JK = 1$. Dotted lines indicate standard error on the test-set performance of the best model in each condition. The greater density of solid symbols in bottom right portion of the figure mean two things: a certain amount of capacity is necessary for good performance, and it is important that the capacity be in the form of model neurons with multiple filters.

3.5 Discussion

Visual system models in our study were more successful at categorizing familiar objects in novel stimuli when their V1-like neurons were able to go beyond the basic linear-nonlinear model and exhibit the range of behaviour found in V1 simple and complex cells. Using a gradient-based method to optimize neuron parameters, the models similar to the classic energy-model (where the firing rate is determined by the sum of squared linear filter responses) demonstrated a superior capacity to generalize from labelled examples of objects. More complex variants on the energy model such as the models of Rust et al. (2005) and Kouh and Poggio (2008) were also better than the basic simple cell model, but brought no consistent advantage over the energy model. The models with squared linear filters were much better at generalizing to new stimuli than the simpler linear-nonlinear models often used in theoretical work.

The most important characteristic of the V1-like neuron model used for image classification was the complex cell-like behaviour, obtained through multiple (from two to at least five) squared linear filters that captured second-order interactions between regions of the receptive field. In terms of learning theory, our results suggest that complex-like models yielded families of functions that were more appropriate for learning to classify objects than linear-nonlinear models (Vapnik, 1995). Large numbers of simple cell models were no substitute for complex cell models because the simple cell models brought a poorer prior over functions for object categorization. The nonlinearities required by the complex-cell models could come from multiple biological sources: feedback from extra-striate (Bredfeldt and Ringach, 2002) and lateral connectivity (Heeger, 1992) could play a role, and dendritic trees have a capacity for nonlinear processing (Haüsser and Mel, 2003; Rhodes, 2008).

One hypothesis for why the complex-like parametrization learnt more quickly is that the sum of squared filters can be more robust to small translations (Adelson and Bergen, 1985). As evidence for this hypothesis, the filters learnt by the most successful models in our study are illustrated in Figure 3.2. The results here are

mixed. The model that was best at discriminating **Shapes** supports the hypothesis; it involves a sum of squared linear filter responses and the filters in the model look like phase-offset gratings that implement angle-invariant selectivity for edges radiating out from various locations near the middle of the image. The model best at discriminating **Digits** supports the hypothesis less strongly; it involves many small Gabor-like oriented edge detectors. These detectors do not have gratings so they would not be robust to displacement of the edge. Squaring of filter responses in these models would make the edge detection robust to changes in polarity. Edges in the digits dataset are almost always close together (the two sides of a pen-stroke) so perhaps polarity invariance is a form of translation invariance in this particular dataset. The model best at discriminating **Toys** offers weak support for the hypothesis; the filters in this model are neither gratings nor edge detectors, and it is not clear (to the authors) how they work. So for at least two of the three datasets in our study (**Shapes** and **Digits**), the filters learnt to implement the computational property (quadrature pairs for translational invariance) that motivated the energy-model. It is not obvious a-priori that for an image x, supervised learning of a transfer function of the form $\sqrt{(ax)^2+(bx)^2}$ should learn phase-offset Gabors and sinusoids for filters a and b. But in at least two of our three datasets that is indeed what happens for the majority of V1-like cells. This finding supports the hypothesis that quadrature pairing for pooling in complex cells is an important computational aspect of low-level feed-forward vision models – when the machinery for that is present, a simple learning algorithm learns to use it that way.

How do our results compare with other published results on the **Digits** dataset (MNIST)? A database of results is online (LeCun, 1998-): chance is 90% error, a linear classifier can get 12% error (LeCun et al., 1998a), K-nearest neighbours 3.09% using an L_2 metric (LeCun, 1998-), a Gaussian-kernel SVM 1.4% error (LeCun, 1998-). With an augmented dataset an SVM can achieve 0.56% error (Decoste and Schölkopf, 2002) and the best deep convolutional neural network achieved 0.39% error (Ranzato et al., 2007). The best digit-classifier in our study (based on classic energy model neurons) scored 1.54% error. It lags behind more sophisticated

machine-learning models, but in this sort of comparison it should be interpreted as a building block for more powerful computer vision models, rather than a complete model in its own right. Our score compares favourably with the standard sigmoidal neural network approach (1.8% error (LeCun, 1998-)), indicating that complex cells can extract more discriminating features than simple cells. Future work will examine the utility of these V1-like models within a hierarchical convolutional architecture that goes further toward replicating the structure of the visual system (Kavukcuoglu et al., 2010; LeCun et al., 1998a; Pinto et al., 2009; Riesenhuber and Poggio, 1999).

We do not know how well a primate visual system would perform in these tasks because for the purpose of comparison, it would be necessary to train visual systems exclusively on these very limited sets of stimuli. Instead, we draw on the approach of rational analysis and appeal to the trivial straightforwardness of these categorization tasks to support our claim that the learning algorithm of the visual system exhibits a preference for functions that are effective in these tasks. The faster learning in the Rust et al. (2005) model agree with the hypothesis that that model's functional priors are closer to the visual system's priors, and that the priors of the other models are further from the visual system's.

Acknowledgements

The authors are immensely grateful for the feedback from Curtis Baker, Paul Cisek, Aaron Courville, Andrea Green, Christopher Pack, Nicole Rust, and Terry Sejnowski, as well as the financial support from NSERC, the Canada Research Chairs, and MITACS.

CHAPTER 4

SLOW FEATURES FOR PRETRAINING COMPLEX CELL NETWORKS

Title	Slow, Decorrelated Features for Pretraining Complex Cell-like Networks
Authors	James Bergstra and Yoshua Bengio
Publication	Proc. of Neural Information Procesing Systems 22: 99–107, 2009.

We introduce a new type of neural network activation function based on recent physiological rate models for complex cells in visual area V1. A single-hidden-layer neural network of this kind of model achieves 1.50% error on MNIST. We also introduce an existing criterion for learning slow, decorrelated features as a pretraining strategy for image models. This pretraining strategy results in orientation-selective features, similar to the receptive fields of complex cells. With this pretraining, the same single-hidden-layer model achieves 1.34% error, even though the pretraining sample distribution is very different from the finetuning distribution. To implement this pretraining strategy, we derive a fast algorithm for online learning of decorrelated features such that each iteration of the algorithm runs in linear time with respect to the number of features.

4.1 Introduction

Visual area V1 is the first area of cortex devoted to handling visual input in the human visual system (Dayan and Abbott, 2001). One convenient simplification in the study of cell behaviour is to ignore the timing of individual spikes, and to look instead at their frequency. Some cells in V1 are described well by a linear filter that has been rectified to be non-negative and perhaps bounded. These so-called *simple cells* are similar to sigmoidal activation functions: their activity (firing frequency) is greater as an image stimulus looks more like some particular linear filter. However, these simple cells are a minority in visual area V1 and the characterization of the remaining cells there (and even beyond in visual areas V2, V4, MT, and so on) is a very active area of ongoing research. *Complex cells* are the next-simplest kind of cell. They are characterized by an ability to respond to narrow bars of light with particular orientations in some region (translation invariance) but to turn off when all those overlapping bars are presented at once. This non-linear response has been modelled by quadrature pairs (Adelson and Bergen, 1985; Dayan and Abbott, 2001): pairs of linear filters with the property that the sum of their squared responses is constant for an input image with particular spatial frequency and orientation (i.e., edges). It has also been modelled by max-pooling across two or more linear filters (Riesenhuber and Poggio, 1999). More recently, it has been argued that V1 cells exhibit a range of behaviour that blurs distinctions between simple and complex cells and between energy models and max-pooling models (Finn and Ferster, 2007; Kouh and Poggio, 2008; Rust et al., 2005).

Another theme in neural modelling is that cells do not react to single images, they react to image sequences. It is a gross approximation to suppose that each cell implements a function from image to activity level. Furthermore, the temporal sequence of images in a video sequence contains a lot of information about the invariances that we would like our models to learn. Throwing away that temporal structure makes learning about objects from images much more difficult. The principle of identifying slowly moving/changing factors in temporal/spatial data has

been investigated by many (Becker and Hinton, 1993; Cadieu and Olshausen, 2009; Hurri and Hyvärinen, 2003; Körding et al., 2004; Wiskott and Sejnowski, 2002) as a principle for finding useful representations of images, and as an explanation for why V1 simple and complex cells behave the way they do. A good overview can be found in (Berkes and Wiskott, 2005).

This work follows the pattern of initializing neural networks with unsupervised learning (*pretraining*) before *finetuning* with a supervised learning criterion. Supervised gradient descent explores the parameter space sufficiently to get low training error on smaller training sets (tens of thousands of examples, like MNIST). However, models that have been pretrained with appropriate unsupervised learning procedures (such as RBMs and various forms of auto-encoders) generalize better (Hinton et al., 2006; Larochelle et al., 2007; Lee et al., 2008b; Ranzato et al., 2008; Vincent et al., 2008). See Bengio (2009) for a comprehensive review and Erhan et al. (2009) for a thorough experimental analysis of the improvements obtained. It appears that unsupervised pretraining guides the learning dynamics in better regions of parameter space associated with basins of attraction of the supervised gradient procedure corresponding to local minima with lower generalization error, even for very large training sets (unlike other regularizers whose effects tend to quickly vanish on large training sets) with millions of examples.

Recent work in the pretraining of neural networks has taken a generative modelling perspective. For example, the Restricted Boltzmann Machine is an undirected graphical model, and training it (by maximum likelihood) as such has been demonstrated to also be a good initialization. However, it is an interesting open question whether a better generative model is necessarily (or even typically) a better point of departure for finetuning. Contrastive divergence (CD) is not maximum likelihood, and works just fine as pretraining. Reconstruction error is an even poorer approximation of the maximum likelihood gradient, and sometimes works better than CD (with additional twists like sparsity or the denoising of Vincent et al. (2008)).

The temporal coherence and decorrelation criterion is an alternative to training

generative models such as RBMs or auto-encoder variants. Recently Mobahi et al. (2009) demonstrated that a slowness criterion regularizing the top-most internal layer of a deep convolutional network during supervised learning helps their model to generalize better. Our model is similar in spirit to pretraining with the semi-supervised embedding criterion at each level (Mobahi et al., 2009; Weston et al., 2008), but differs in the use of decorrelation as a mechanism for preventing trivial solutions to a slowness criterion. Whereas RBMs and denoising autoencoders are defined for general input distributions, the temporal coherence and decorrelation criterion makes sense only in the context of data with slowly-changing temporal or spatial structure, such as images, video, and sound.

In the same way that simple cell models were the inspiration for sigmoidal activation units in artificial neural networks and validated simple cell models, we investigate in artificial neural network classifiers the value of complex cell models. This paper builds on these results by showing that the principle of temporal coherence is useful for finding initial conditions for the hidden layer of a neural network that biases it toward better generalization in object recognition. We introduce temporal coherence and decorrelation as a pretraining algorithm. Hidden units are initialized so that they are invariant to irrelevant transformations of the image, and sensitive to relevant ones. In order for this criterion to be useful in the context of large models, we derive a fast online algorithm for decorrelating units and maximizing temporal coherence.

4.2 Algorithm

4.2.1 Slow, Decorrelated Feature Learning Algorithm

Körding et al. (2004) introduced a principle (and training criterion) to explain the formation of complex cell receptive fields. They based their analysis on the complex-cell model of Adelson and Bergen (1985), which describes a complex cell as a pair of half-rectified linear filters whose outputs are squared and added together and then a square root is applied to that sum.

Suppose x is an input image and we have F complex cells $h_1, ..., h_F$ such that $h_i = \sqrt{(u_i \cdot x)^2 + (v_i \cdot x)^2}$. Körding et al. (2004) showed that by minimizing the following cost,

$$\mathscr{L}^{(\text{K2004})} = \alpha \sum_{i!=j} \frac{\text{Cov}_t(h_i, h_j)^2}{\text{Var}(h_i)\text{Var}(h_j)} + \sum_t \sum_i \frac{(h_{i,t} - h_{i,t-1})^2}{\text{Var}(h_i)}, \tag{4.1}$$

over consecutive natural movie frames (with respect to model parameters), the filters u_i and v_i of each complex cell form local Gabor filters whose phases are offset by about 90 degrees, like the sine and cosine curves that implement a Fourier transform.

The criterion in Equation 4.1 requires a *batch* minimization algorithm because of the variance and covariance statistics that must be collected. This makes the criterion too slow for use with large datasets. At the same time, the size of the covariance matrix is quadratic in the number of features, so it is computationally expensive (perhaps prohibitively) to apply the criterion to train large numbers of features.

4.2.1.1 Online Stochastic Estimation of Covariance

This section presents an algorithm for approximately minimizing $\mathscr{L}^{(\text{K2004})}$ using an online algorithm whose iterations run in linear time with respect to the number of features. One way to apply the criterion to large or infinite datasets is by estimating the covariance (and variance) from consecutive minibatches of N movie frames. Then the cost can be minimized by stochastic gradient descent.

We used an exponentially-decaying moving average to track the mean of each feature over time, $\bar{h}_i(t) = \rho \bar{h}_i(t-1) + (1-\rho) h_i(t)$. For good results, ρ should be chosen so that the estimates change very slowly. We used a value of $1.0 - 5.0 \times 10^{-5}$. Then we estimated the variance of each feature over a minibatch as in Eqn. 4.2.

$$\text{Var}(h) \approx \frac{1}{N-1} \sum_{\tau=t}^{t+N-1} (h_i(t) - \bar{h}_i(t))^2. \tag{4.2}$$

With this mean and variance, we computed normalized features for each minibatch as in Eqn 4.3.

$$z_i(t) = (h_i(t) - \bar{h}_i(t))/\sqrt{\mathrm{Var}(h) + 10^{-10}}. \tag{4.3}$$

Letting Z denote an $F \times N$ matrix with N columns of F normalized feature values, we estimate the correlation between features h_i by the covariance in these normalized features: $C(t) = \frac{1}{N} Z(t)Z(t)'$. We can now write down $L(t)$, in Eqn. 4.4 a minibatch-wise approximation to Eqn. 4.1:

$$L(t) = \alpha \sum_{i!=j} C_{ij}^2(t) + \sum_{\tau=0}^{N-1} \sum_i (z_i(t+\tau) - z_i(t+\tau-1))^2. \tag{4.4}$$

The time complexity of evaluating $L(t)$ from Z using this expression is $O(FFN + NF)$. In practice we use small minibatches and our model has lots of features, so the fact that the time complexity of the algorithm is quadratic in F is troublesome. However, this value can be computed exactly in time linear in F. The key observation is that the sum of the squared elements of C can be computed from the $N \times N$ *Gram* matrix $G(t) = Z(t)'Z(t)$.

$$\begin{aligned}
&\sum_{i=1}^{F} \sum_{j=1}^{F} C_{ij}^2(t) = \mathrm{Tr}(C(t)C(t)) \\
&= \frac{1}{N^2} \mathrm{Tr}(Z(t)Z(t)'Z(t)Z(t)') = \frac{1}{N^2} \mathrm{Tr}(Z(t)'Z(t)Z(t)'Z(t)) \\
&= \frac{1}{N^2} \mathrm{Tr}(G(t)G(t)) = \frac{1}{N^2} \mathrm{Tr}(G(t)G(t)') \\
&= \frac{1}{N^2} \sum_{k=1}^{N} \sum_{l=1}^{N} G_{kl}^2(t) \doteq \frac{1}{N^2} |Z(t)'Z(t)|^2
\end{aligned}$$

Subtracting the C_{ii}^2 terms from the sum of all squared elements lets us rewrite Eqn. 4.4 in a way that suggests the linear-time implementation (Eqn. 4.5).

$$L(t) = \frac{\alpha}{N^2} \left(|Z(t)Z'(t)|^2 - \sum_{i=1}^{F} (\sum_{\tau=1}^{N} z_i(\tau)^2)^2 \right) + \frac{1}{N-1} \sum_{\tau=1}^{N-1} \sum_{i=1}^{F} (z_i(\tau) - z_i(\tau-1))^2 \tag{4.5}$$

The time complexity of computing $L(t)$ using Equation 4.5 from $Z(t)$ is $O(NNF)$. The sum of squared correlations is still the most expensive term, but for the case where $N << F$, this expression makes $L(t)$'s computation linear in F. Considering that each iteration treats N training examples, the per-training-example cost of this algorithm can be seen as $O(NF)$. In implementation, an additional factor of two in runtime can be obtained by only computing half of the Gram matrix G, which is symmetric.

4.2.2 Complex Cell Activation Function

Recently, Rust et al. (2005) have argued that existing models, such as that of Adelson and Bergen (1985) cannot account for the variety of behaviour found in visual area V1. Some complex cells behave like simple cells to some extent and vice versa; there is a continuous range of simple to complex cells. They put forward a similar but more involved expression that can capture the simple and complex cells as special cases, but ultimately parameterizes a larger class of cell-response functions (Eq. 4.6).

$$a + \frac{\beta \left(\max(0, wx)^2 + \sum_{i=1}^{I} (u^{(i)} x)^2 \right)^{\zeta} - \delta \left(\sum_{j=1}^{J} (v^{(j)} x)^2 \right)^{\zeta}}{1 + \gamma \left(\max(0, wx)^2 + \sum_{i=1}^{I} (u^{(i)} x)^2 \right)^{\zeta} + \varepsilon \left(\sum_{j=1}^{J} (v^{(j)} x)^2 \right)^{\zeta}} \tag{4.6}$$

The numerator in Eq. 4.6 describes the difference between an *excitation* term and a *shunting inhibition* term. The denominator acts to normalize this difference. Parameters $w, u^{(i)}, v^{(j)}$ have the same shape as the input image x, and can be thought of as image filters like the first layer of a neural network or the codebook of a sparse-coding model. The parameters $a, \beta, \delta, \gamma, \varepsilon, \zeta$ are scalars that control the range and shape of the activation function, given all the filter responses. The numbers I and J of quadratic filters required to explain a particular cellular response were on the order of 2-16.

We introduce the approximation in Equation 4.7 because it is easier to learn by gradient descent. We replaced the max operation with a softplus$(x) = \log(1 + e^x)$

function so that there is always a gradient on w and b, even when $wx+b$ is negative. We fixed the scalar parameters to prevent the system from entering regimes of extreme non-linearity. We fixed $\beta, \delta, \gamma, \varepsilon$ to 1, and a to 0. We chose to fix the exponent ζ to 0.5 because Rust et al. (2005) found that values close to 0.5 offered good fits to cell firing-rate data. Future work might look at choosing these constants in a principled way or adapting them; we found that these values worked well. The range of this activation function (as a function of x) is a connected set on the $(-1,1)$ interval. However, the whole $(-1,1)$ range is not always available, depending on the parameters. If the inhibition term is always 0 for example, then the activation function will be non-negative.

$$\frac{\sqrt{\log(1+e^{wx+b})^2+\sum_{i=1}^{I}(u^{(i)}x)^2}-\sqrt{\sum_{j=1}^{J}(v^{(j)}x)^2}}{1.0+\sqrt{\log(1+e^{wx+b})^2+\sum_{i=1}^{I}(u^{(i)}x)^2}+\sqrt{\sum_{j=1}^{J}(v^{(j)}x)^2}} \tag{4.7}$$

4.3 Results

Classification results were obtained by adding a logistic regression model on top of the features learnt, and treating the resulting model as a single-hidden-layer neural network. The weights of the logistic regression were always initialized to zero.

All work was done on 28x28 images (MNIST-sized), using a model with 300 hidden units. Each hidden unit had one linear filter w, a bias b, two quadratic excitatory filters u_1, u_2 and two quadratic inhibitory filters v_1, v_2. The computational cost of evaluating each unit was thus five times the cost of evaluating a normal sigmoidal activation function of the form $\tanh(w'x+b)$.

4.3.1 Random Initialization

As a baseline, our model parameters were initialized to small random weights and used as the hidden layer of a neural network. Training this randomly-initialized model by stochastic gradient descent yielded test-set performance of 1.56% on

MNIST.

The filters learnt by this procedure looked somewhat noisy for the most part, but had low-frequency trends. For example, some of the quadratic filters had small local Gabor-like filters. We believe that these phase-offset pairs of Gabor-like functions allow the units to implement some shift-invariant response to edges with a specific orientation (Fig. 4.1).

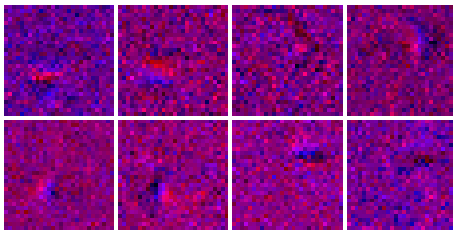

Figure 4.1: Four of the three hundred activation functions learnt by training our model from random initialization to perform classification. Top row: the red and blue channels are the two quadratic filters of the excitation term. Bottom row: the red and blue channels are the two quadratic filters of the shunting inhibition term. Training yields locally orientation-selective edge filters, opposite-orientation edges are inhibitory.

4.3.2 Pretraining with Natural Movies

Under the hypothesis that the matched Gabor functions (see Fig. 4.1) allowed our model to generalize better across slight translations of the image, we appealed to a pretraining process to initialize our model with values better than random noise.

We pretrained the hidden layer according to the online version of the cost in Eq. 4.5, using movies (MIXED-movies) made by sliding a 28 x 28 pixel window across large photographs. Each of these movies was short (just four frames long) and ten movies were used in each minibatch ($N = 40$). The sliding speed was sampled uniformly between 0.5 and 2 pixels per frame. The sliding direction was sampled uniformly from 0 to 2π. The sliding initial position was sampled uniformly from image coordinates. Any sampled movie that slid off of the underlying image was

rejected. We used two photographs to generate the movies. The first photograph was a grey-scale forest scene (resolution 1744x1308). The second photograph was a tiling of 100x100 MNIST digits (resolution 2800x2800). As a result of this procedure, digits are not at all centred in MIXED-movies: there might be part of a '3' in the upper-left part of a frame, and part of a '7' in the lower right.

The shunting inhibition filters (v_1, v_2) learnt after five hundred thousand movies (fifty thousand iterations of stochastic gradient descent) are shown in Figure 4.2. The filters learn to implement orientation-selective, shift-invariant filters at different spatial frequencies. The filters shown in figure 4.2 have fairly global receptive fields, but smaller more local receptive fields were obtained by applying ℓ_1 weight-penalization during pretraining. The α parameter that balances decorrelation and slowness was chosen manually on the basis of the trained filters. We were looking for a diversity of filters with relatively low spatial frequency. The excitatory filters learnt similar Gabor pairs but the receptive fields tended to be both smaller (more localized) and lower-frequency. Finetuning this pretrained model with a learning rate of 0.003 with ℓ_1 weight decay of 10^{-5} yielded a test error rate of 1.34% on MNIST.

4.3.3 Pretraining with MNIST movies

We also tried pretraining with videos whose frames follow a similar distribution to the images used for finetuning and testing. We created MNIST movies by sampling an image from the training set, and moving around (translating it) according to a Brownian motion. The initial velocity was sampled from a zero-mean normal distribution with std-deviation 0.2. Changes in that velocity between each frame were sampled from zero-mean normal distribution with std-deviation 0.2. Furthermore, the digit image in each frame was modified according to a randomly chosen elastic deformation, as in Loosli et al. (2007). As before, movies of four frames were created in this way and training was conducted on minibatches of ten movies ($N = 4 * 10 = 40$). Unlike the MNIST frames in MIXED-movies, the frames of MNIST-movies contain a single digit that is approximately centred.

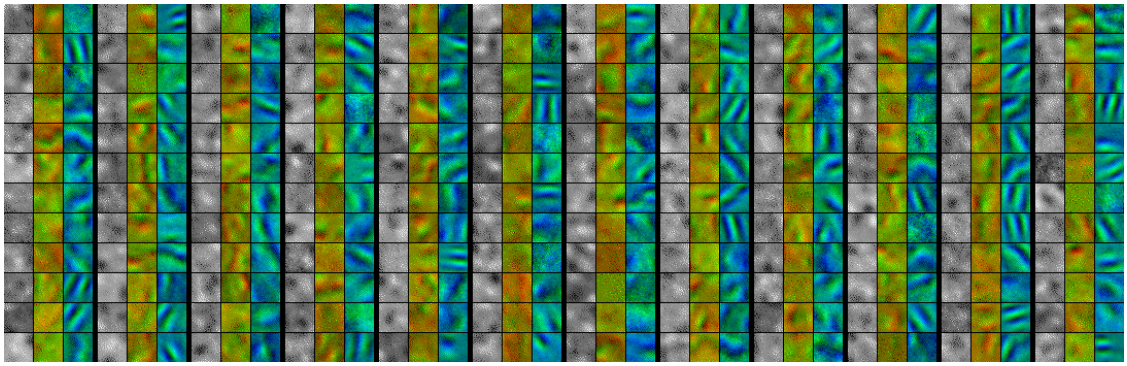

Figure 4.2: Filters from some of the units of the model, pretrained on small sliding image patches from two large images. The features learn to be direction-selective for moving edges by approximately implementing windowed Fourier transforms. These features have global receptive field, but become more local when an ℓ_1 weight penalization is applied during pretraining. Excitatory filters looked similar, but tended to be more localized and with lower spatial frequency (fewer, shorter, broader stripes). Columns of the figure are arranged in triples: linear filter w in grey, $u^{(1)}, u^{(2)}$ in red and green, $v^{(1)}, v^{(2)}$ in blue and green.

The activation functions learnt by minimizing Equation 4.5 on these MNIST movies were qualitatively different from the activation functions learnt from the MIXED movies. The inhibitory weights (v_1, v_2) learnt from MNIST movies are shown in 4.3. Once again, the inhibitory weights exhibit the narrow red and green stripes that indicate edge-orientation selectivity. But this time they are not parallel straight stripes, they follow contours that are adapted to digit edges. The excitation filters u_1, u_2 were also qualitatively different. Instead of forming localized Gabor pairs, some formed large smooth blob-like shapes but most converged toward zero. Finetuning this pretrained model with a learning rate of 0.003 with ℓ_1 weight decay of 10^{-5} yielded a test error rate of 1.37 % on MNIST.

4.4 Discussion

The results on MNIST compare well with many results in the literature. A single-hidden layer neural network of sigmoidal units can achieve 1.8% error by training from random initial conditions, and our model achieves 1.5% from random

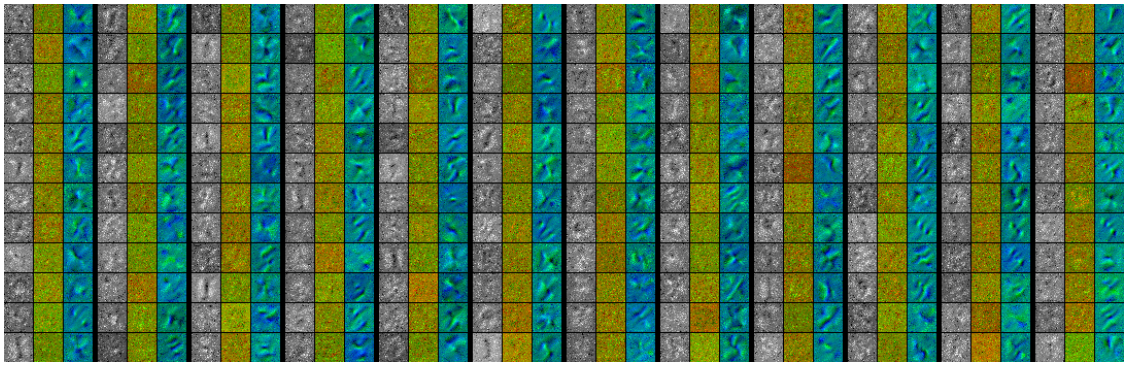

Figure 4.3: Filters of our model, pretrained on movies of centred MNIST training images subjected to Brownian translation. The features learn to be direction-selective for moving edges by approximately implementing windowed Fourier transforms. The filters are tuned to the higher spatial frequency in MNIST digits, as compared with the natural scene. Columns of the figure are arranged in triples: linear filter w in grey, $u^{(1)}, u^{(2)}$ in red and green, $v^{(1)}, v^{(2)}$ in blue and green.

initial conditions. A single-hidden layer sigmoidal neural network pretrained as a denoising auto-encoder (and then finetuned) can achieve 1.4% error on average, and our model is able to achieve 1.34% error from many different finetuned models (Erhan et al., 2009). Gaussian SVMs trained just on the original MNIST data achieve 1.4%; our pretraining strategy allows our single-layer model be better than Gaussian SVMs (Decoste and Schölkopf, 2002). Deep learning algorithms based on denoising auto-encoders and RBMs are typically able to achieve slightly lower scores in the range of $1.2-1.3\%$ (Erhan et al., 2009; Hinton et al., 2006). The best convolutional architectures and models that have access to enriched datasets for finetuning can achieve classification accuracies under 0.4% (Ranzato et al., 2007). In future work, we will explore strategies for combining these methods and with our decorrelation criterion to train deep networks of models with quadratic input interactions. We will also look at comparative performance on a wider variety of tasks.

4.4.1 Transfer Learning, the Value of Pretraining

To evaluate our unsupervised criterion of slow, decorrelated features as a pretraining step for classification by a neural network, we finetuned the weights obtained after ten, twenty, thirty, forty, and fifty thousand iterations of unsupervised learning. We used only a small subset (the first 100 training examples) from the MNIST data for supervised learning to focus on the importance of pretraining. The results are listed in Table 4.I. Training from random weights initial led to 23.1 % error. The value of pretraining is evident right away: after two unsupervised passes over the MNIST training data (100K movies and 10K iterations), the weights have been initialized better. Finetuning the weights learnt on the MIXED-movies led to test error rate of 21.2%, and finetuning the weights learnt on the MNIST-movies led to a test error rate of 19.0%. Further pretraining offers a diminishing marginal return, although after ten unsupervised passes through the training data (500K movies) there is no evidence of over-pretraining. The best score (20.6%) on MIXED-movies occurs at both eight and ten unsupervised passes, and the best score on MNIST-movies (18.4%) occurs after eight. A larger test set would be required to make a strong conclusion about a downward trend in test set scores for larger numbers of pretraining iterations. The results with MNIST-movies pretraining are slightly better than MIXED-movies but these results suggest strong transfer learning: the videos featuring digits in random locations and natural image patches are almost as good for pretraining as compared with videos featuring images very similar to those in the test set.

4.4.2 Slowness, Normalization, and Binary Activations

Somewhat counter-intuitively, the slowness criterion requires movement in the features h. Suppose a feature h_i has activation levels that are normally distributed around 0.1 and 0.2, but the activation at each frame of a movie is independent of previous frames. Since the features has a small variance, then the normalized feature z_i will oscillate in the same way, but with unit variance. This will cause

Table 4.I: Generalization error (% error) from 100 labelled MNIST examples after pretraining on MIXED-movies and MNIST-movies.

Pretraining Dataset	Number of pretraining iterations ($\times 10^4$)					
	0	1	2	3	4	5
MIXED-movies	23.1	21.2	20.8	20.8	20.6	20.6
MNIST-movies	23.1	19.0	18.7	18.8	18.4	18.6

$z_i(t) - z_i(t-1)$ to be relatively high, and for our slowness criterion not to be well satisfied. In this way the lack of variance in h_i can actually make for a relatively *fast* normalized feature z_i rather than a slow one.

However, if h_i has activation levels that are normally distributed around .1 and .2 for some image sequences and around .8 and .9 for other image sequences, the marginal variance in h_i will be larger. The larger marginal variance will make the oscillations between .1 and .2 lead to much smaller changes in the normalized feature $z_i(t)$. In this sense, the slowness objective can be maximally satisfied by features $h_i(t)$ that take near-minimum and near-maximum values for most movies, and never transition from a near-minimum to a near-maximum value during a movie.

When training on multiple short videos instead of one continuous one, it is possible for large changes in normalized-feature-activation never [or rarely] to occur during a video. Perhaps this is one of the roles of saccades in the visual system: to suspend the normal objective of temporal coherence during a rapid widespread change of activation levels.

4.4.3 Eigenvalue Interpretation of Decorrelation Term

What does our unsupervised cost mean? One way of thinking about the decorrelation term (first term in Eq. 4.1) which helped us to design an efficient algorithm for computing it, is to think of it as flattening the eigen-spectrum of the correlation matrix of our features h (over time). It is helpful to rewrite this cost in terms of

normalized features: $z_i = \frac{h_i - \bar{h}_i}{\sigma_i}$, and to consider that we sum over all the elements of the correlation matrix including the diagonal.

$$\sum_{i!=j} \frac{\mathrm{Cov}_t(h_i, h_j)^2}{\mathrm{Var}(h_i)\mathrm{Var}(h_j)} = 2 \sum_{i=1}^{F-1} \sum_{j=i+1}^{F} \mathrm{Cov}_t(z_i, z_j)^2 = \left(\sum_{i=1}^{F} \sum_{j=1}^{F} \mathrm{Cov}_t(z_i, z_j)^2 \right) - F$$

If we use C to denote the matrix whose i, j entry is $\mathrm{Cov}_t(z_i, z_j)$, and we use $U'\Lambda U$ to denote the eigen-decomposition of C, then we can transform this sum over $i! = j$ further.

$$\begin{aligned}
&(\sum_{i=1}^{F} \sum_{j=1}^{F} \mathrm{Cov}_t(z_i, z_j)^2) - F = \mathrm{Tr}(C'C) - F = \mathrm{Tr}(CC) - F \\
&= \mathrm{Tr}(U'\Lambda U U'\Lambda U) - F = \mathrm{Tr}(UU'\Lambda UU'\Lambda) - F = \sum_{k=1}^{F} \Lambda_k^2 - F
\end{aligned}$$

We can interpret the first term of Eq. 4.1 as penalizing the squared eigenvalues of the covariance matrix between features in a normalized feature space (z as opposed to h), or as minimizing the squared eigenvalues of the correlation matrix between features h.

4.5 Conclusion

We have presented an activation function for use in neural networks that is a simplification of a recent rate model of visual area V1 complex cells. This model learns shift-invariant, orientation-selective edge filters from purely supervised training on MNIST and achieves lower generalization error than conventional neural nets.

Temporal coherence and decorrelation has been put forward as a principle for explaining the functional behaviour of visual area V1 complex cells. We have described an online algorithm for minimizing correlation that has linear time complexity in the number of hidden units. Pretraining our model with this unsupervised

criterion yields even lower generalization error: better than Gaussian SVMs, and competitive with deep denoising auto-encoders and 3-layer deep belief networks. The good performance of our model compared with poorer approximations of V1 is encouraging machine learning research inspired by neural information processing in the brain. It also helps to validate the corresponding computational neuroscience theories by showing that these neuron activations and unsupervised criteria have value in terms of learning.

Acknowledgments

This research was performed thanks to funding from NSERC, MITACS, and the Canada Research Chairs.

CHAPTER 5

THE SPIKE AND SLAB RBM

Title	A Spike and Slab Restricted Boltzmann Machine
Authors	Aaron Couville, James Bergstra, and Yoshua Bengio
Publication	Accepted to AISTATS, 2011.

We introduce the *spike and slab* Restricted Boltzmann Machine, characterized by having both a real valued vector, *the slab*, and a binary variable, *the spike*, associated with each unit in the hidden layer. The model possesses some practical properties such as being amenable to Block Gibbs sampling as well as being capable of generating similar latent representations of the data to the recently introduced mean and covariance Restricted Boltzmann Machine. We illustrate how the spike and slab Restricted Boltzmann Machine achieves competitive performance on a standard object recognition task with natural image data.

5.1 Introduction

The prototypical Restricted Boltzmann Machine (RBM) is a Markov random field with a bipartite graph structure that divides the model variables into two layers: a visible layer consisting of binary variables representing the data, and a hidden (or latent) layer consisting of the latent binary variables. The bipartite structure excludes connections between the variates (or units) within each layer so that the units within the hidden layer are conditionally independent given the units of the visible layer, and the visible layer units are conditionally independent given the hidden layer units. This pair of conditionally factorial distributions permits a simple block Gibbs sampler, alternating between the dual conditionals $P(\textsf{visible layer} \mid \textsf{hidden layer})$ and $P(\textsf{hidden layer} \mid \textsf{visible layer})$. The ability to sample simply and efficiently from the RBM forms the basis for effective learning algorithms such as contrastive divergence (CD, Carreira-Perpiñan and Hinton, 2005; Hinton, 2002) and stochastic maximum likelihood (SML, Tieleman, 2008; Younes, 1998).

While the RBM has proved effective in a range of tasks and data domains (Chen and Murray, 2003; Larochelle and Bengio, 2008; Nair and Hinton, 2009; Salakhutdinov et al., 2007; Sutskever et al., 2009; Taylor and Hinton, 2009), it has not been as successful in modelling continuous multivariate data, and natural images in particular (Ranzato and Hinton, 2010). The most popular approach to modelling continuous observations within the RBM framework was the so-called Gaussian RBM (GRBM), defined such that the conditional distribution of the visible layer given the hidden layer is Gaussian with the conditional mean being parametrized by a set of weights and the *binary* hidden unit values, and a fixed covariance. Thus the GRBM can be viewed as a Gaussian mixture model with the number of components being exponential in the number of hidden units, while the number of parameters (the weights) being only linear in the number of hidden units.

The GRBM has proved unsatisfactory as a model of natural images, as the trained features typically do not represent sharp edges that occur at object bound-

aries and lead to latent representations that are not particularly useful features for classification tasks (Ranzato and Hinton, 2010). Ranzato and Hinton (2010) have argued that the failure of the GRBM to adequately capture the statistical structure apparent in natural images stems from the exclusive use of the model capacity to capture the conditional mean at the expense of the conditional covariance. While we agree that the GRBM provides a poor covariance model, we suggest that this deficiency has more to do with the binary nature of the hidden layer units than with the model's devotion to capturing the conditional mean.

Our perspective on the GRBM motivates us to reconsider the strategy of modelling continuous-valued inputs with strictly binary latent variables, and leads us to the spike and slab Restricted Boltzmann Machine (ssRBM). Like many RBM variants, the spike and slab RBM is restricted to a bipartite graph structure between two types of nodes. The visible layer units are modelled as real valued variables as in the GRBM approach. Where our model departs from other similar methods is in the definition of the hidden layer latent variables. We model these as the element-wise product of a real valued vector with a binary vector, i.e., each hidden unit is associated with a binary *spike* variable and the real vector valued *slab* variable. The name *spike and slab* is inspired from terminology in the statistics literature (Mitchell and Beauchamp, 1988), where the term refers to a prior consisting of a mixture between two components: the spike, a discrete probability mass at zero; and the slab, a density (typically uniformly distributed) over a continuous domain.

In this paper, we show how the introduction of the slab variables to the GRBM leads to an interesting new RBM. By marginalizing out the slab variables, the conditional distribution of the spike variables given the input is very similar to the corresponding conditional of the recently introduced covariance RBM (cRBM) (Ranzato et al., 2010a). On the other hand, conditional on the spike variables, the ssRBM slab variables and input are jointly Gaussian and form conditionals with diagonal covariance matrices. Thus, unlike the cRBM or its extension the mean-covariance RBM (mcRBM), the ssRBM is amenable to simple and efficient Gibbs sampling.

This property of the ssRBM makes the model an excellent candidate as a building block for the development of more sophisticated models such as the Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009).

As we develop the model, we show that with multidimensional slab variables, feature "sum" pooling becomes a natural part of the model. In the experiments, we illustrate how maximum likelihood training of the ssRBM yields filters that capture natural image properties such as sharp edges. We also show how the model exhibits "disentangling" of colour and edge features when trained on natural image patches and how the ssRBM can learn state-of-the-art features for the CIFAR-10 object classification dataset (Krizhevsky, 2009).

5.2 The Inductive Bias of the GRBM

Before delving into the development of the ssRBM, we first elaborate on our perspective that the failure of the GRBM to model natural images is due to the use of binary hidden units. We argue this case by comparing the GRBM to a standard Gaussian factor model with a Gaussian-distributed latent vector, $x \in \mathbb{R}^N$, and a Gaussian conditional distribution over the observations, $v \in \mathbb{R}^D$, given the latent variable. That is to say, $x \sim \mathcal{N}(0, \Sigma_x)$ and $v|x \sim \mathcal{N}(Wx, \sigma_v \mathbf{I})$, where W is a matrix $(D \times N)$ of weights. Under this model, variations in a single element x_i reflect covariance within the observation vector along the direction (in the input or v space) of $W_{:,i}$. Indeed marginalizing out the latent variables, we are left with the marginal distribution over the observation vector: $p_x(v) \sim \mathcal{N}(0, \sigma_v \mathbf{I} + W \Sigma_x W^T)$. Note that the weights W that parametrize the conditional mean serve also to parametrize the marginal covariance. The GRBM is different from the Gaussian factor model in a number of important ways, but most relevant for our purposes, the GRBM replaces the real valued latent variables of the factor model with binary variables. If we replace the real valued x in the factor model with simple binary variables h, the equivalence between parameterizing the conditional mean and parameterizing the marginal covariance breaks down. Instead of a single Gaussian with covariance

$\sigma_v \mathbf{I} + W \Sigma_x W^T$, the marginal distribution $p(v)$ becomes the mixture of Gaussians: $p_h(v) = \sum_h P(v) \mathcal{N}(Wh, \sigma_v \mathbf{I})$.

This change from a real variable x to a binary variable h has an impact on the inductive bias of the model and consequently an impact on the suitability of the model to a particular data domain. Both the zero-mean Gaussian $p_x(v)$ and the mixture model $p_h(v)$ exhibit a preference (in the sense of higher probability density) for data distributed along the directions of the columns of their respective weight matrices. However, if the statistical structure of the data is such that density should be relatively invariant to overall scaling of v, then the inductive bias resulting from the binary h may be inappropriate. Figure 5.1 illustrates how the discrete mixture components in $p_h(v)$ are ill-suited to model natural images, where some of the most significant determiners of the norm of the data vector $\|v\|_2$ are the illumination conditions of the scene and the image contrast. Variation in contrast often bears little relevance to typical tasks of interest such as object recognition or scene understanding. This perspective on the GRBM, and especially its comparison to the standard Gaussian factor model, motivates us to consider alternatives to strictly binary latent variables and leads us to the spike and slab RBM.

5.3 The Spike and Slab RBM

Let the number of hidden units be N, and the dimensionality of the visible vector be D: $v \in \mathbb{R}^D$. The ith hidden unit ($1 \leq i \leq N$) is associated with a binary

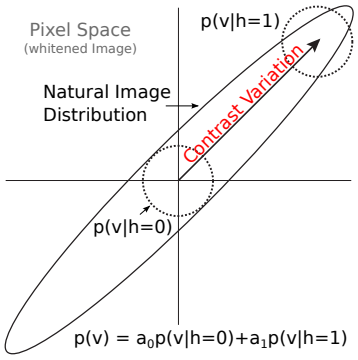

Figure 5.1: The GRBM exhibits significant sensitivity to variation in contrast.

spike variable: $h_i \in \{0,1\}$ and a real valued vector $s_i \in \mathbb{R}^K$, pooling over the K features.[1] The energy function for one example is:

$$E(v,s,h) = \frac{1}{2}v^T \Lambda v - \sum_{i=1}^{N}\left(v^T W_i s_i h_i - \frac{1}{2}s_i^T \alpha_i s_i + b_i h_i\right), \tag{5.1}$$

where W_i refers to the ith weight matrix of size $D \times K$, the b_i are the biases associated with each of the spike variables h_i, and α_i and Λ are diagonal matrices that penalize large values of $\|s_i\|_2^2$ and $\|v\|_2^2$ respectively. We will consider a joint probability distribution over v, $s = [s_1, \ldots, s_N]$ and $h = [h_i, \ldots, h_N]$ of the form:

$$p(v,s,h) = \frac{1}{Z}\exp\{-E(v,s,h)\} \times \mathbb{U}(v;R) \tag{5.2}$$

where, Z is the partition function that assures that $p(v,s,h)$ is normalized and $\mathbb{U}(v;R)$ represents a distribution that is uniform over a ball radius R, centred at the origin, that contains all the training data, i.e., $R > \max_t \|v_t\|_2$ (t indexes over training examples). The region of the visible layer space outside the ball has zero probability under the model. This restriction to a finite domain guarantees that the partition function Z remains finite. We can think of the distribution presented in equations 5.2 and 5.1, as being associated with the bipartite graph structure of the RBM with the distinction that the hidden layer is composed of an element-wise product of the vectors s and h.

With the joint distribution thus defined, we now turn to deriving the set of conditional distributions $p(v \mid s,h)$, $p(s \mid v,h)$, $P(h \mid v)$ and $p(v \mid h)$ from which we can gain some insight into the properties of the ssRBM. The strategy we will adopt is to derive the conditionals neglecting the $\mathbb{U}(v;R)$ factor, then during sampling we can correct for the omission via rejection sampling. This turns out to be very efficient as the number of rejections is expected to be very low as we will discuss later in section 5.4.

Let us first consider the conditional distribution $p(v \mid s,h)$. Taking into account the bounded domain of v, we have $p(v \mid s,h,\|v\|_2 > R) = 0$ and:

1. It is perhaps more natural to consider a scalar s_i, i.e., $K = 1$; however generalizing to vector valued s_i allows us to naturally implement a form of "sum" pooling.

$$
\begin{aligned}
p(v \mid s,h,\|v\|_2 \leq R) &= \frac{1}{p(s,h)}\frac{1}{Z}\exp\{-E(v,s,h)\} \\
&= \frac{1}{B}\mathcal{N}\left(\Lambda^{-1}\sum_{i=1}^{N} W_i s_i h_i \ , \ \Lambda^{-1}\right),
\end{aligned}
$$

where B is determined by integrating the Gaussian $\mathcal{N}\left(\Lambda^{-1}\sum_{i=1}^{N} W_i s_i h_i \ , \ \Lambda^{-1}\right)$ over the ball $\|v\|_2 \leq R$. By isolating all terms involving v, the remaining terms are constant with respect to v and therefore the conditional distribution $p(v \mid s,h)$ has the form of a simple (truncated) Gaussian distribution and since the off-diagonal terms of the covariance are all zero, sampling from this Gaussian is straightforward, when using rejection sampling to exclude v outside the bounded domain. For convenience, we will adopt the notation $p^*(v \mid s,h)$ to refer to the un-truncated Gaussian distribution associated with $p(v \mid s,h)$; i.e., $p^*(v \mid s,h) = \mathcal{N}\left(\Lambda^{-1}\sum_{i=1}^{N} W_i s_i h_i \ , \ \Lambda^{-1}\right)$

It is instructive to consider what happens if we do not assume we know s, i.e., considering the form of the distribution $p(v \mid h)$ where we marginalize out s:

$$
\begin{aligned}
p(v \mid h,\|v\|_2 \leq R) &= \frac{1}{P(h)}\frac{1}{Z}\int \exp\{-E(v,s,h)\}\ ds \\
&= \frac{1}{B}\mathcal{N}\left(0 \ , \ \left(\Lambda - \sum_{i=1}^{N} h_i W_i \alpha_i^{-1} W_i^T\right)^{-1}\right) \qquad (5.3)
\end{aligned}
$$

The last equality holds only if the covariance matrix $\left(\Lambda - \sum_{i=1}^{N} h_i W_i \alpha_i^{-1} W_i^T\right)^{-1}$ is positive definite. By marginalizing over the "slab" variates, s, the visible vector v remains (truncated) Gaussian-distributed, however the parametrization has changed significantly as a function of h. The distribution $p^*(v \mid s,h)$ uses h with s to parametrize the conditional mean, whereas in the case of $p^*(v \mid h)$, h parametrizes the conditional covariance. Another critical difference between these two distributions over v is that the covariance matrix of the Gaussian $p^*(v \mid h)$ is not diagonal. As such, sampling from $p^*(v \mid h)$ is potentially computationally intensive for large v as it would require a matrix inverse for every weight update. Fortunately, we will

have no need to sample from $p^*(v \mid h)$.

We now turn to the conditional $p(s_i \mid v,h)$. The derivation is analogous to that leading to Eq. 5.3. The conditional $p(s \mid v,h)$ is Gaussian-distributed:

$$p(s \mid v,h) = \prod_{i=1}^{N} \mathcal{N}\left(h_i \alpha_i^{-1} W_i^T v \ , \ \alpha_i^{-1}\right) \tag{5.4}$$

Here again, we see that the conditional distribution over s given v and h possess a diagonal covariance enabling simple and efficient sampling of s from this conditional distribution. The form of $p(s \mid v,h)$ indicates that, given $h_i = 1$, the expected value of s_i is linearly dependent of v.

Similar to $p(v \mid h)$, the distribution $p(h \mid v)$ is obtained by marginalizing out the slab variable s:

$$\begin{aligned} P(h_i = 1 \mid v) &= \frac{1}{p(v)} \frac{1}{Z_i} \int \exp\{-E(v,s,h)\} \ ds \\ &= \text{sigm}\left(\frac{1}{2} v^T W_i \alpha_i^{-1} W_i^T v - b_i\right), \end{aligned} \tag{5.5}$$

where **sigm** represents a logistic sigmoid. As with the conditionals $p(v \mid s,h)$ and $p(s \mid v,h)$, the distribution of h given v factorizes over the elements of h. As a direct consequence of the marginalization of s, the influence of v on $P(h_i \mid v)$ is controlled by a term quadratic in $v^T W_i$, meaning that h_i is active when v has a significant projection onto the direction W_i (which is one of particular variance among training data).

A choice of data representations: The spike and slab RBM is somewhat unusual in that the use of dual latent variables, one continuous, and one binary, offers us a choice of data representations, to be used in the particular task at hand. One option is to marginalize over s, and use the binary h or its expectation $P(h \mid v)$ as the data representation. Another option is to use $[s_1 h_1, \ldots, s_N h_N]$ or $[\|s_1\| h_1, \ldots, \|s_N\| h_N]$ or the corresponding expectations. These options possess the property that, for active units, the model representation is equivariant to the intensity of the input variable (within the bounded domain). This is a property shared with the rectified linear units of Nair and Hinton (2010), and is thought to

be potentially beneficial in a range of vision tasks as it offers superior robustness to variations in image intensity.

5.4 ssRBM Learning and Inference

As is typical of RBM-style models, learning and inference in the ssRBM is dependent on the ability to efficiently draw samples from the model via Markov chain Monte Carlo (MCMC). Inspection of the conditionals $P(h \mid v)$, $p(v \mid h)$, $p(s \mid v,h)$ and $p(v \mid s,h)$ reveals some important property of the ssRBM model. First, let us consider the standard RBM sampling scheme of iterating between $P(h \mid v)$ and $p(v \mid h)$ with s marginalized out. Sampling from $P(h \mid v)$ is straightforward, as equation 5.5 indicates that the h_i are all independent given v. Under the assumption of a positive definite covariance matrix, the conditional distribution $p(v \mid h)$ is multivariate Gaussian with non-diagonal covariance: $\left(\Lambda - \sum_{i=1}^{N} h_i W_i \alpha_i^{-1} W_i^T\right)^{-1}$. Thus sampling from $p(v \mid h)$ requires the inversion of the covariance matrix with every weight update. For large input dimensionality D, this presents a challenging setting for learning. Fortunately, we need not sample from $p(v \mid h)$ directly, instead we can instantiate the slab variable s by sampling from $p(s \mid h,v)$ and then, given these s samples and the h sampled from $P(h \mid v)$, we can sample v from the conditional $p(v \mid s,h)$. Both these conditionals are Gaussian with diagonal covariance leading to simple and efficient sampling.

Taken all together the triplet $P(h \mid v)$, $p(s \mid v,h)$ and $p(v \mid s,h)$ form the basis of a block-Gibbs sampling scheme that allows us to sample efficiently from the ssRBM. Whenever a sample of v falls outside the ball $\|v\|_2 \leq R$, we reject and resample from the conditional $p(v \mid s,h)$. The data likelihood gradient is

$$\begin{aligned} \frac{\partial}{\partial \theta_i}\left(\sum_{t=1}^{T} \log p(v_t)\right) = & - \sum_{t=1}^{T} \left\langle \frac{\partial}{\partial \theta_i} E(v_t,s,h) \right\rangle_{p(s,h|v_t)} \\ & + T \left\langle \frac{\partial}{\partial \theta_i} E(v,s,h) \right\rangle_{p(v,s,h)}, \end{aligned} \tag{5.6}$$

i.e., of the same form as for a standard RBM, only with the expectations over $p(s,h \mid$

v_t) in the "clamped" condition, and over $p(v,s,h)$ in the "unclamped" condition. In training, we follow the stochastic maximum likelihood algorithm (also known as persistent contrastive divergence, Tieleman, 2008; Younes, 1998), i.e., performing only one or few updates of an MCMC chain between each parameter update.

The expectation of the gradient with respect to W_i in the "clamped" condition (also called the positive phase) is:

$$\left\langle \frac{\partial}{\partial W_i} E(v_t,s,h) \right\rangle_{p(s,h|v_t)} = -v_t \left(\mu_{i,t}^+\right)^T \hat{h}_{i,t}^+. \tag{5.7}$$

Here $\hat{h}_{i,t}^+ = p(h_i \mid v_t)$ and $\mu_{i,t}^+$ is the mean of the Gaussian density $p(s_i|h_i = 1, v_t)$. In the "unclamped" condition (negative phase) the expectation of the gradient with respect to W_i is given by:

$$\left\langle \frac{\partial}{\partial W_i} E(v,s,h) \right\rangle_{p(v,s,h)} \approx \frac{1}{M} \sum_{m=1}^{M} -\tilde{v}_m (\mu_{i,m}^-)^T \hat{h}_{i,m}^-. \tag{5.8}$$

Where $\hat{h}_{i,m}^- = p(h_i \mid \tilde{v}_m)$ and $\mu_{i,m}^-$, is the mean of the Gaussian density $p(s_i \mid h_i = 1, \tilde{v}_m)$. The $\tilde{v}_m$ are samples drawn from the model via Gibbs sampling. The expectation of the gradient with respect to b_i is identical to that of the GRBM. Finally, the expectation of the gradient with respect to Λ is given by:

$$\left\langle \frac{\partial}{\partial \Lambda} E(v_t,s,h) \right\rangle_{p(s,h|v_t)} = \frac{1}{2} v_t^T v_t \tag{5.9}$$

$$\left\langle \frac{\partial}{\partial \Lambda} E(v,s,h) \right\rangle_{p(v,s,h)} \approx \frac{1}{M} \sum_{m=1}^{M} \frac{1}{2} \tilde{v}_m^T \tilde{v}_m \tag{5.10}$$

One could also imagine updating α to maximize likelihood; however in our experiments we simply treated α as a hyper-parameter.

As previously discussed, without the $\mathbb{U}(v;R)$ term in the joint density, the spike and slab model is not parametrized to guarantee that the model constitutes a well defined probability model with a finite partition function. To draw samples from the model, we rely on a rejection sampling scheme based on $\mathbb{U}(v;R)$. However, during training, we instead rely on a very important property of the likelihood gradient to suppress samples from the model that are drawn in regions of the data

space unsupported by nearby data examples. As the parameters are updated, the model may approach instability. If this occurs, negative phase or "unclamped" samples are naturally drawn to the direction of the instability (i.e., outside the range of the training data) and through their influence act to return the model to a locally stable region of operation. Due to this stabilizing property of learning, we actually do not include the $\mathbb{U}(v;R)$ term to the joint likelihood during learning. Practically, training the model is straightforward provided the model is initialized in a stable regime of the parameter space. For example, the values of α and Λ must be sufficiently large to at least offset the initial values of W. We also use a decreasing learning rate that also helps maintain the model in a stable region of the parameter space. Training this way also ensures that the natural parametrization of the ssRBM (excluding the $\mathbb{U}(v;R)$) is almost always sufficient to ensure stability during sampling and renders our rejection sampling strategy highly efficient.

5.5 Comparison to Previous Work

There exist a number of papers that aim to address the issue of modelling natural images in the RBM context. The most relevant of these are the Product of Student's T-distribution (PoT) model (Welling et al., 2003) and the mean and covariance Restricted Boltzmann Machine (mcRBM) (Ranzato and Hinton, 2010). However before reviewing these models and their connections to the ssRBM, we note that the idea of building Boltzmann Machines with products of binary and continuous-valued variables was discussed in Williams (1993), Zemel et al. (1993), and Freund and Haussler (1994). We also note that the covariance structure of the ssRBM conditional $p(v \mid h)$ (equation 5.3) is essentially identical to the product of probabilistic principal components analysis (PoPPCA) model (Williams, 2001) with components corresponding to the ssRBM weight vectors associated with the active hidden units ($h_i = 1$).

5.5.1 Product of Student's T-distributions

The product of Student's T-distributions model (Welling et al., 2003) is an energy-based model where the conditional distribution over the visible units conditioned on the hidden variables is a multivariate Gaussian (non-diagonal covariance) and the complementary conditional distribution over the hidden variables given the visibles are a set of independent Gamma distributions. The PoT model is similar to our model in that it characterizes the covariance of real-valued inputs with real valued hidden units, but in the case of the PoT model, the real valued hidden units are Gamma-distributed rather than Gaussian-distributed as is the case for the ssRBM.

The most significant difference between the ssRBM and the PoT model is how they parametrize the covariance of the multivariate Gaussian over the visible units ($p(v \mid h)$ in the case of the ssRBM, equation 5.3). While the ssRBM characterizes the covariance as $\left(\Lambda - \sum_{i=1}^{N} h_i W_i \alpha_i^{-1} W_i^T\right)^{-1}$, the PoT model parametrized the conditional covariance as $\left(\sum_{i=1}^{N} u_i W_i W_i^T\right)^{-1}$, where the u_i are the Gamma-distributed latent variables. The PoT latent variables use their activation to maintain constraints, decreasing in value to allow variance in the direction of the corresponding weight vector. The spike and slab h_i variables use their activation to pinch the precision matrix along the direction specified by the corresponding weight vector. The two models diverge when the dimensionality of the hidden layer exceeds that of the input. In the over-complete setting, sparse activation with the ssRBM parametrization permits significant variance (above the nominal variance given by Λ^{-1}) only in the select directions of the sparsely activated h_i. This is a property the ssRBM shares with sparse coding models (Grosse et al., 2007; Olshausen and Field, 1997) where the sparse latent representation also encodes directions of variance above a nominal value. An over-complete PoT model has a different interpretation: with an over-complete set of constraints, variation of the input along a particular direction would require decreasing potentially all constraints with positive projection in that direction.

5.5.2 The Mean and Covariance RBM

One recently introduced and particularly successful approach to modelling real valued data is the mean and covariance RBM. The mcRBM is a restricted Boltzmann machine designed to explicitly model both the mean and covariance of elements of the input. The mcRBM combines a variant of the earlier covariance RBM (cRBM) model (Ranzato et al., 2010a) with a GRBM to capture the conditional "mean". Because of some surprising similarities between the cRBM and the ssRBM, we will review the cRBM in some detail.

We take the number of cRBM hidden units to be N_c: $h^c \in \{0,1\}^{N_c}$, and the dimensionality of the visible vector to be D: $v \in \mathbb{R}^D$. The cRBM model is defined via the energy function:

$$E^c(v,h^c) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} P_{ki}h_i^c\left(v^T C_{:,k}\right)^2 - \sum_{i=1}^{N} b_i^c h_i^c, \tag{5.11}$$

where P is a pooling matrix with non-positive elements($p \in \mathbb{R}^{K\times N}$), N is the number of hidden units, $C_{:,k}$ is the weight vector k ($C \in \mathbb{R}^{D\times K}$) and b^c is a vector of biases. Defining the energy function in this way allows one to derive the pair of conditionals for h and v respectively as:

$$\begin{aligned} P(h_i^c = 1 \mid v) &= \text{sigm}\left(\frac{1}{2}\sum_{k=1}^{K} P_{ki}h_i^c\left(v^T C_{:,k}\right)^2 - b_i^c\right), \\ p(v \mid h^c) &= \mathcal{N}\left(0, \left(C\ \text{diag}(Ph^c)C^T\right)^{-1}\right), \end{aligned} \tag{5.12}$$

where $\text{diag}(v)$ is the diagonal matrix with vector v in its diagonal. That is, the conditional Gaussian distribution possess a non-diagonal covariance.

In relation to the ssRBM, the first thing to note about the cRBM is the similarity of the conditional for the binary latent variable, $P(h \mid v)$ in the case of the ssRBM (equation 5.5) and $P(h^c \mid v)$ in the case of the cRBM. Simplifying both models to pool over a single variable (setting the P matrix to the negative identity in the case of the cRBM and $K = 1$ in the ssRBM), both conditionals contain a $\frac{1}{2}(v^T W)^2$ term (with $C \equiv W$) and a constant bias. Remarkably, this occurs despite the two models

sharing relatively little in common at the level of the energy function.

Despite the similarity in the conditional distribution over the binary latent variables, the two models diverge in their expressions for the complementary conditions over the visible variable v given the binary latents (comparing equations 5.3 and 5.12). While the ssRBM parametrizes the covariance as $\left(\Lambda - \sum_{i=1}^{N} h_i W_i \alpha_i^{-1} W_i^T\right)^{-1}$; the cRBM parametrizes the covariance as $\left(C\ \mathrm{diag}(Ph^c)C^T\right)^{-1}$. Similar to the PoT model, the cRBM encodes the conditional covariance as a series of constraints to be actively enforced. As is the case for the PoT model, we suggest that this form of parametrization is not well suited to heavily over-complete models.

Despite different parameterizations of the conditional covariance, the ssRBM and the cRBM share the property that the conditional distribution over v given their respective binary latent variables is multivariate Gaussian with a non-diagonal covariance. In the ssRBM, we have recourse to a simple diagonal-covariance Gaussian conditional over v by instantiating the slab variables s, but there is no equivalent recourse for the cRBM. As a result, the cRBM and the mcRBM are not amenable to the kind of block Gibbs sampling available to the ssRBM and to more standard RBMs (a large matrix inversion would be required for each Gibbs step). In training the cRBM, samples are drawn using hybrid Monte Carlo (HMC) (Neal, 1994). As an MCMC sampler, HMC has been shown to be very effective for some problems, but it suffers from a relatively large number of hyper-parameters that must be tuned to yield well-mixing samples from the target distribution.

The mcRBM combines a GRBM with a cRBM such that there are two kinds of hidden units, mean units h^m and covariance units h^c. The combined energy function of the mcRBM is given by:

$$E(v, h^c, h^m) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} P_{ki} h_i^c \left(\frac{v^T}{\|v\|}\frac{C_{:,k}}{\|C_k\|}\right)^2$$
$$-\sum_{i=1}^{N} b_i^c h_i^c + \frac{1}{2}v^T v - \sum_{j=1}^{M} v^T W_{:,j} h_j^m - \sum_{j=1}^{M} b_j^m h_j^m$$

The mcRBM is not entirely equivalent to the combination of a cRBM and a GRBM,

as its energy function includes a normalization of both the C_k weight vectors and the visible vector (to increase the robustness of the model to large contrast variations in the input image).

In deriving the conditionals for the ssRBM, we saw that by manipulating how we treat the slab variables s, we could fluidly move between modelling the conditional mean and modelling the conditional covariance. From this perspective it is revealing to think about the combination of the GRBM with the cRBM in the mcRBM. One can think about an equivalent model, within the spike and slab framework, where we take a subset of the ssRBM latent units and marginalize over the corresponding slab variables s – these unit would encode the conditional covariance. With the remaining units we model the equivalent conditional mean by imposing the constraint $s_i = \mathbf{1}$.

5.6 Experiments

We have run simulations to illustrate three key ideas related to the ssRBM model: (a) it learns appealing filters to model natural images, (b) the spike variables are meaningfully used in a trained model, and (c) the latent image representation induced by the ssRBM makes the ssRBM a drop-in upgrade of the similar GRBM and cRBM models on CIFAR-10 image-labelling, and is competitive with the more complicated mcRBM.

5.6.1 Filters

The ssRBM learnt qualitatively similar filters in the pooled and un-pooled models, but the pooling induced interesting structure to the set of filters.

Figure 5.2 illustrates the filters learnt by an un-pooled ($K = 1$) ssRBM from a large number (one million) of PCA-whitened 8x8 RGB image patches drawn from the TinyImages dataset (Torralba et al., 2008). PCA-whitening retained 99% of the variance with 74 dimensions. These filters were obtained by stochastic maximum likelihood learning with the learning rate set to 10^{-4} for 20 000 training iterations

using minibatches of size 128. After 20 000 iterations the learning rate was reduced in inverse proportion to the iteration number. No sparsification or regularization was applied to the activations or model parameters. α was fixed to 1.5, the bias was initialized to -1, the weights were initialized from a zero-mean Gaussian with variance 10^{-4}.

Figure 5.3 illustrates the effect of pooling $K = 9$ scale variables s with each h. The pinwheel-like pattern was obtained by sharing columns $W_{:,i}$ between pools using the sort of topographic map used in Ranzato and Hinton (2010). Clean backgrounds in each filter were obtained by applying a small (10^{-4}) ℓ_1 penalty to the filter weights. All filters were brought into play by applying a small (.2) ℓ_1 penalty pushing each unit h_i to have a marginal mean of .1. The topographic map down-weighted the effective magnitude of each W column, so the initial range and learning rate on W were raised accordingly.

5.6.2 The Effect of Spike Variables

Figure 5.4 illustrates the effect of the binary spike variables (h). The effect of h_i is to suppress the influence of filter $W_{:,i}$ when the filter response is weak. Once h has been inferred from an observation v it induces a Gaussian conditional joint distribution $p(s, v \mid h)$ as well as a Gaussian conditional marginal $p(v \mid h)$ in which the covariance is determined by the filters that were unusually active.Figure 5.4 shows

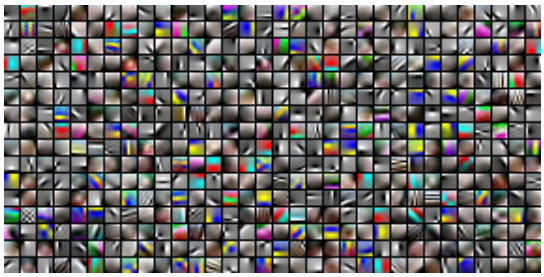

Figure 5.2: Filters learnt by the unpooled ssRBM when applied to PCA-whitened 8x8 colour image patches. Note how some filters care about colour while others surprisingly do not, achieving a form of disentangling of colour information from shape information.

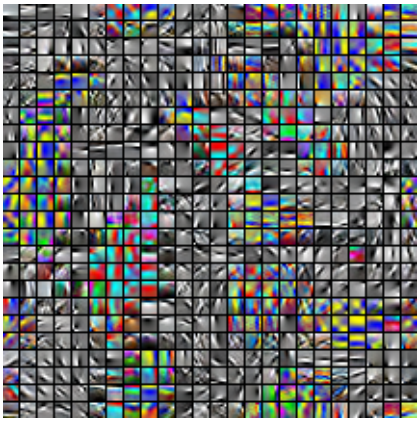

Figure 5.3: Filters learnt by a pooled spike and slab RBM with topographically overlapping pools applied to PCA-whitened 8x8 color image patches. Pooling was done across 3x3 groups ($K = 9$) units s giving rise to a degree of continuity across the set of filters. Again, colour and grey-level filters nicely separate.

that the spike variables are indeed often 0, and eliminating potential directions of covariance in the conditional marginal.

5.6.3 Learning Features for Classification

To evaluate the latent variables of the ssRBM as features for object classification we adopted the testing protocol of Ranzato and Hinton (2010) which looked at performance on CIFAR-10. CIFAR-10 comprises 40 000 training images, 10 000 validation images, and 10 000 test images. The images are 32-by-32 pixel RGB images. Each image is labelled with one of ten object categories (aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) according to the most prominent object in the image. We produced image features from an ssRBM model trained on patches using the same procedure as Ranzato and Hinton (2010) - a 7-by-7 grid of (overlapping) 8-by-8 pixel patches was used to extract a 7-by-7 grid of mean-values for h. The ssRBM had N h variables, so the concatenation of the h vectors from each grid location yielded $49N$ features. We classified this feature vector using logistic regression, and we also experimented with back-propagating the error gradient into the ssRBM parameters and finetuning this "feature extractor" as if it and the classifier together were a single neural network.

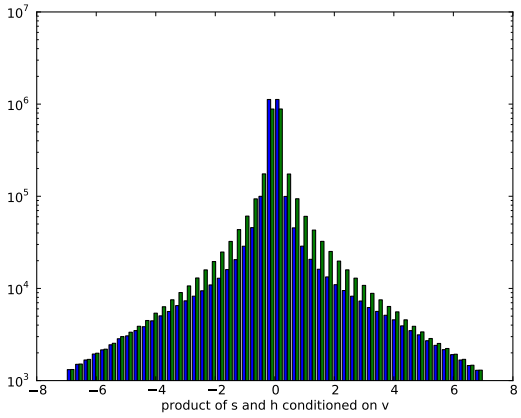

Figure 5.4: The spike and slab effect - in green is the marginal (over a large number of images) distribution over all s_i variables given $h_i = 1$, and in blue is the marginal distribution over all $s_i h_i$ products. The vertical axis is a log-scaled frequency of occurrence. When slab variable s would have been small anyway, the spike variable h tends to be zero, and thereby to make their product exactly zero.

We optimized hyper-parameters for this task by drawing 200 hyper-parameter assignments randomly from a grid, performing 50 000 and 200 000 unsupervised training iterations, measuring classification error, and sorting all these unsupervised models by the validation set performance of the $P(h|v)$ feature vector. The random grid included variations in the number of unsupervised learning iterations (50K, 200K), learning rate (.0003, .0001), initial Λ (10, 15, 20), number of latent h variables N (100, 200,400), number of pooled s variables K per h (1,2,3), initial range for W (.05, .1, .2), initial bias on each h_i (-5, -4, -3), target sparsity for each h (.05, .1, .2), weight of sparsity regularization (0, .1, .2, .4). The initial value of α was fixed to 10.5. The best results with and without finetuning of the ssRBM weight matrix are given in Table 5.I along with selected other results from the literature.

We also experimented with "mean" units as in Ranzato and Hinton (2010) by adding $s_i h_i$ pairs in which the s_i were fixed to 1. Reusing the best-performing hyper-parameters, we simply added 81 mean units and repeated the training procedure. Interestingly, as in Ranzato and Hinton (2010) we found that these additional mean units improved the performance of the model for classification beyond what was

Table 5.I: The performance of the pooled and unpooled ssRBM models relative to other models in the literature for CIFAR-10. 95% confidence intervals are given for each score assuming the official test set size of ten thousand. The mssRBM model is an ssRBM with 81 of the s units fixed to 1. Finetuned models were trained for classification as convolutional neural networks with a single hidden layer.

Model	Classification Rate (%)
mssRBM (finetuned)	69.9 ±0.9
mssRBM	68.7 ±0.9
mcRBM	68.2 ± 0.9
ssRBM (finetuned)	69.2 ±0.9
ssRBM	67.6 ±0.9
cRBM (900 factors)	64.7 ±0.9
cRBM (225 factors)	63.6 ±0.9
GRBM [2]	59.7 ±1.0

found by adding additional normal (unclamped) hidden units. This result, that a hidden layer consisting of a mix of mean and pooled units is better than either one alone, suggests that models with heterogeneous latent states (layers) represent an interesting direction for future work. Ranzato and Hinton (2010) also demonstrate superior classification performance by adding binary RBM layers (as in Hinton et al. (2006)) on top of the latent binary variables of the mcRBM (70.7% for two extra layers, 71.0% for five extra layers). We expect the ssRBM performance would also show improve with additional layers.

5.7 Discussion

In this paper we introduce a new RBM model we call a *spike and slab RBM*. The model is characterized by having a binary spike variable and a continuous slab variable associated with each hidden unit. The introduction of the slab variables allows the model to naturally capture covariance information while simultaneously maintaining simple and efficient inference via a Gibbs sampling scheme. The ssRBM is competitive with the current state-of-the-art on the CIFAR-10 object categorization task.

Despite the similarity in the conditional distributions over the hidden binary variables between the ssRBM and the cRBM, there are a number of important distinctions. First, the ssRBM is amenable to Gibbs sampling whereas when sampling from the cRBM one must resort to hybrid Monte Carlo (HMC). While HMC is a practical algorithm for sampling in the RBM framework, the simplicity of Gibbs makes the ssRBM a more attractive option as a building block for more ambitious models such as the deep Boltzmann machine (Salakhutdinov and Hinton, 2009) and time-series models (Taylor and Hinton, 2009). Another difference between the ssRBM and the cRBM is that the ssRBM induces sparse real valued representations of the data. In our limited experiments using this data representation, we have not found it to be superior to using only $P(h \mid v)$, however recent work Nair and Hinton (2010) has demonstrated the importance of sparse real valued outputs in achieving superior classification performance.

As discussed previously, without any restriction on either the visible layer domain or the binary hidden unit combinations, the energy of the spike and slab model is not guaranteed to be bounded and consequently is not guaranteed to define a valid probability distribution. In practice this is fairly easily dealt with by imposing either a bounded domain on v, as we have done, or by applying a global penalty that is flat in the region of the training data and outside that region grows sufficiently fast to overcome any negative growth arising from the energy function (i.e., the term $-\sum_{i=1}^{N} v^T W_i s_i h_i$). As an alternative, one could restrict the covariance of $p(v \mid h)$ to remain positive definite and reject patterns of hidden unit activations that violate this constraint. Under the mixture model interpretation of the RBM, this approach may be interpreted as zeroing out the mixture components that violate the requirement that the mixture components be individually normalizable. Finally, as our title insinuates, the ssRBM introduced here is simply one member of a family of spike and slab RBMs. Different choices within the energy function can lead to max-pooling behaviour among the filters or non-negative *slab* variables. Further study is required to fully explore the model space.

CHAPTER 6

THE μ-SPIKE-AND-SLAB RBM

Title	Unsupervised Models of Images by Spike and Slab RBMs
Authors	Aaron Couville, James Bergstra, and Yoshua Bengio
Publication	Submitted to ICML, 2011.

The *spike and slab* Restricted Boltzmann Machine (RBM) is defined by having both a real valued *slab* variable and a binary *spike* variable associated with each unit in the hidden layer. In this paper, we generalize and extend the spike and slab RBM in two ways: (i) to include a non-zero mean in the distribution over observed variables when conditioning on the binary spike variables, and (ii) to constrain the parameters of the model so that all conditionals associated with the model are always well defined – a guarantee absent from the original spike and slab RBM. The inclusion of (i) improves the performance of the spike and slab RBM as a feature learner and allows it to achieves competitive performance on the CIFAR-10 image classification task. Surprisingly we find that each of several types of constraints (ii) hurt classification performance – the model prefers to operate closer to the regime of ill-definedness (sometimes actually crossing over, apparently quite harmlessly) than our constraints permit. When trained in a convolutional configuration, the μ-spike-and-slab RBM generates more realistic samples than similar methods, which demonstrate that the model can capture the broad statistical structure of natural images.

6.1 Introduction

Recently, there has been considerable interest in the problem of unsupervised learning of features for supervised tasks in natural image domains. Approaches based on unsupervised *pretraining* followed by either whole-model *finetuning* or simply the linear classification of features dominate in benchmark tasks such as CIFAR-10 (Krizhevsky, 2009).

One of most popular energy-based modelling paradigms for unsupervised feature learning is the Restricted Boltzmann Machine (RBM). An RBM is a Markov random field with a bipartite graph structure consisting of a visible layer and a hidden layer. The bipartite structure excludes connections between the variables within each layer so that the latent variables are conditionally independent given the visible variables and vice versa. The factorial nature of these conditional distributions enables efficient Gibbs sampling which forms the basis of the most widespread RBM learning algorithms such as contrastive divergence (Hinton, 2002) and stochastic maximum likelihood (Tieleman, 2008).

The proposed spike and slab RBM (ssRBM) (Courville et al., 2010) departs from other similar RBM-based models in the way the hidden layer latent units are defined. They are modelled as the element-wise product of a real valued vector with a binary vector, i.e., each hidden unit is associated with a binary *spike* variable and a real-valued *slab* variable. The name *spike and slab* is inspired from terminology in the statistics literature (Mitchell and Beauchamp, 1988), where the term refers to a prior consisting of a mixture between two components: the spike, a discrete probability mass at zero; and the slab, a density (typically uniformly distributed) over a continuous domain.

In this paper, we introduce a generalization of the ssRBM model, which we refer to as the μ-ssRBM. Relative to the original ssRBM, the μ-ssRBM includes additional terms in the energy function which give extra modelling capacity. One of the additional terms allows the model to form a conditional distribution of the spike variables (by marginalizing out the slab variables, given an observation) that

is similar to the corresponding conditional of the recently introduced mean covariance RBM (mcRBM) (Ranzato and Hinton, 2010) and mPoT model (Ranzato et al., 2010b). Conditional on both the observed and spike variables, the μ-ssRBM slab variables and input are jointly Gaussian with diagonal covariance matrix; conditional on both the spike and slab variables, the observed variables are Gaussian with diagonal covariance. Thus, unlike the mcRBM or the more recent mPoT model, the μ-ssRBM is amenable to simple and efficient Gibbs sampling. This property of the ssRBM makes the model an excellent candidate as a building block for the development of more sophisticated models such as the Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009).

One potential drawback of the spike and slab RBM is the lack of a guarantee that the resulting model constitutes a valid density over the whole real space in which the data exist. This issue is an important aspect of the analysis and variants explored in this paper. We show several strategies that guarantee all conditionals are well defined by adding energy terms to the μ-ssRBM. However, we find that loosening the constraint experimentally yields better models.

6.2 The μ-Spike-and-Slab RBM

The μ-ssRBM describes the interaction between three random vectors: the visible vector v representing the observed data, the binary "spike" variables h and the real-valued "slab" variables s. The ith hidden unit is associated both with an element h_i of the binary vector and with an element s_i of the real-valued variable. Suppose there are N hidden units: $h \in [0,1]^N$ and a visible vector of dimension D: $v \in \mathbb{R}^D$. The μ-ssRBM model is defined via the energy function

$$\begin{aligned} E(v,s,h) = &- \sum_{i=1}^{N} v^T W_i s_i h_i + \frac{1}{2} v^T \left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i \right) v \\ &+ \frac{1}{2} \sum_{i=1}^{N} \alpha_i s_i^2 - \sum_{i=1}^{N} \alpha_i \mu_i s_i h_i - \sum_{i=1}^{N} b_i h_i + \sum_{i=1}^{N} \alpha_i \mu_i^2 h_i, \end{aligned} \tag{6.1}$$

in which W_i denotes the ith weight vector ($W_i \in \mathbb{R}^D$), each b_i is a scalar bias associated with h_i, each α_i is a scalar that penalizes large values of s_i^2, and Λ is a diagonal matrix that penalizes large values of $\|v\|_2^2$. In comparison with the original ssRBM (Courville et al., 2010), the μ-ssRBM energy function includes three additional terms. Firstly, the $1/2\ v^T \left(\sum_{i=1}^N \Phi_i h_i\right) v$ term, with non-negative diagonal matrices $\Phi_i,\ i \in [1,N]$, establishes an h-dependent quadratic penalty on v. Secondly, associated with each slab variable is a mean parameter μ – from which the μ-ssRBM takes its name. Finally, the $\sum_{i=1}^N \alpha_i \mu_i^2 h_i$ term acts as an additional bias term for the h_i, which we include to simplify the parametrization of the conditionals. In addition to offering additional flexibility to the μ-ssRBM to model the statistics of natural images, the inclusion of the parameters $\mu = [\mu_1, \ldots, \mu_N]$ and $\Phi = [\Phi_1, \ldots, \Phi_N]$ also allows us to derive constraints on the model that ensure that the model remains well-behaved over the entire data domain of $\mathbb{R}^D$.

The joint probability distribution over v, $s = [s_1, \ldots, s_N]$ and $h = [h_1, \ldots, h_N]$ is defined as:

$$p(v,s,h) = \frac{1}{Z} \exp\{-E(v,s,h)\} \tag{6.2}$$

where Z is the normalizing partition function. We can think of the distribution presented in Eqns. 6.1 and 6.2 as associated with the standard RBM bipartite graph structure with the distinction that the hidden layer is composed of an element-wise product of the random vectors s and h.

To gain insight into the μ-ssRBM model, we will look at the conditional distributions $p(v \mid s,h)$, $p(s \mid v,h)$, $P(h \mid v)$ and $p(v \mid h)$. First, we consider the conditional $p(v \mid s,h)$:

$$p(v \mid s,h) = \frac{1}{p(s,h)} \frac{1}{Z} \exp\{-E(v,s,h)\} \tag{6.3}$$

$$= \mathcal{N}\left(v;\ C_{v|s,h} \sum_{i=1}^{N} W_i s_i h_i,\ C_{v|s,h}\right) \tag{6.4}$$

where $C_{v|s,h} = \left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i\right)^{-1}$. The conditional distribution of v given both s and h is Gaussian-distributed with mean $C_{v|s,h} \sum_{i=1}^{N} W_i s_i h_i$ and covariance $C_{v|s,h}$. Since Λ and Φ_i are diagonal (for $i \in [1,N]$), the covariance matrix of $p(v \mid s,h)$ is also diagonal. Eqn. 6.4 shows the role played by the Φ_i in augmenting the precision with the activation of h_i. Indeed each hidden unit contributes a component not only to the mean proportional to $W_i s_i$, but also to the global scaling of the conditional mean (through Φ_i).

The conditional over the slab variables s given the spike variables h and the visible units v is given by:

$$p(s \mid v,h) = \prod_{i=1}^{N} \mathcal{N}\left(v;\ \left(\alpha_i^{-1} v^T W_i + \mu_i\right) h_i\ ,\ \alpha_i^{-1}\right). \tag{6.5}$$

As was the case with the conditional $p(v \mid s,h)$, deriving the conditional $p(s \mid v,h)$ from the joint distribution in Eqn. 6.2 reveals a Gaussian distribution with diagonal covariance. Eqn. 6.5 also shows how the mean of the slab variable s_i, given $h_i = 1$, is linearly dependent on v, and as the precision $\alpha_i \to \infty$, s_i converges in probability to μ_i.

Marginalizing out the slab variables s yields the traditional RBM conditionals $p(v \mid h)$ and $p(h \mid v)$. The conditional $p(v \mid h)$ is also Gaussian,

$$\begin{aligned} p(v \mid h) &= \frac{1}{P(h)} \frac{1}{Z} \int \exp\{-E(v,s,h)\}\ ds \\ &= \mathcal{N}(C_{v|h} \sum_{i=1}^{N} W_i \mu_i h_i, C_{v|h}) \end{aligned} \tag{6.6}$$

where $C_{v|h} = \left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i - \sum_{i=1}^{N} \alpha_i^{-1} h_i W_i W_i^T\right)^{-1}$ and this covariance matrix $C_{v|h}$ is positive definite. Note that the covariance is not obviously parametrized to guarantee that it is positive definite. In Section 6.3, we will discuss strategies to ensure that $C_{v|h}$ be positive definite via constraints on Λ and Φ. In marginalizing over s, the visible vector v remains Gaussian-distributed, but the parametrization has changed. While the distribution $p(v \mid s,h)$ uses h with s to parametrize the condi-

tional mean with a corresponding diagonal covariance, the conditional $p(v \mid h)$ uses h to parametrize a non-diagonal covariance matrix (due to the $\sum_{i=1}^{N} \alpha_i^{-1} h_i W_i W_i^T$ term) and a conditional mean mediated by μ.

Note that according to Eqn. 6.6, the conditional mean of v given h and principal axis of conditional covariance are generally in a similar direction, and if $\left(\Lambda + \sum_{i=1}^{N} \Phi_i h_i\right)$ is a scalar matrix (equivalent to $\textbf{scalar} \times \textbf{Identity}$) the two vectors are in exactly the same direction. This is an important aspect of the inductive bias of the model. Having the principal component of the conditional covariance in the same direction as the mean has the property of $p(v \mid h)$ being maximally invariant to changes in $\|v\|_2$. This is a desirable property for a model of natural images where $\|v\|_2$ is particularly sensitive to illumination conditions and image contrast levels – factors that are often irrelevant to tasks of interest such as object classification.

The final conditional that we will consider is the distribution over the latent spike variables h given the visible vector, $P(h \mid v) = \prod_i^N P(h_i \mid v)$ and

$$\begin{aligned} P(h_i = 1 \mid v) &= \frac{1}{p(v)} \frac{1}{Z_i} \int \exp\{-E(v,s,h)\} \; ds \\ &= \sigma\left(\frac{1}{2}\alpha_i^{-1}(v^T W_i)^2 + v^T W_i \mu_i - \frac{1}{2} v^T \Phi_i v + b_i\right), \end{aligned} \tag{6.7}$$

where σ represents a logistic sigmoid. As with the conditionals $p(v \mid s,h)$ and $p(s \mid v,h)$, the distribution of h given v factorizes over the elements of h. Eqn. 6.7 shows the interaction between three data-dependent terms. The first term, $\frac{1}{2}\alpha_i^{-1}(v^T W_i)^2$, is the contribution due to the variance in s about its mean (note the scaling with α_i^{-1}) and appears in the sigmoid as a result of marginalizing out s. This term is always non-negative, so it always acts to increase $P(h_i \mid v)$. Countering this tendency to activate h_i is the other term quadratic in v, $-\frac{1}{2} v^T \Phi_i v$, that is always a non-positive contribution to the sigmoid argument. In addition to these two quadratic terms, there is the term $v^T W_i \mu_i$ whose behaviour mimics the data-dependent term in the analogous GRBM version of the conditional distribution over h: $P_{\mathrm{G}}(h_i \mid v) = \sigma\left(v^T W_i + b_i\right)$.

Another perspective on the behaviour of $p(h_i \mid v)$ as a function of v is gained by considering an alternative arrangement of the terms:

$$P(h_i{=}\,1 \mid v) = \sigma(\hat{b}_i - \frac{1}{2}(v - \xi_{v|h_i})^T C_{v|h_i}^{-1}(v - \xi_{v|h_i})), \tag{6.8}$$

in which $C_{v|h_i} = \left(\Phi_i - \alpha_i^{-1} W_i W_i^T\right)^{-1}$, $\xi_{v|h_i} = C_{v|h_i} W_i \mu_i$ and $\hat{b}_i$ is a re-parameterization of the bias that incorporates the remainder of the completion of the square. It is evident from this form of $P(h_i = 1 \mid v)$ that, in the event that the matrix $C_{v|h_i}$ is positive definite, then $P(h_i \mid v)$ reaches its maximum when $v = C_{v|h_i} W_i \mu_i$. We can easily confirm that the tail behaviour, as v departs from its maximum, is Gaussian: for $x \to \infty$,

$$\sigma(-x^2) = \frac{\exp(-x^2)}{1 + \exp(-x^2)} \to \exp(-x^2). \tag{6.9}$$

We will look to take advantage of the μ-ssRBM relationship between Φ_i and $\alpha_i^{-1} W_i W_i^T$ when we consider ways to constrain the covariance of $p(v \mid h)$ to be positive definite in Section 6.3.

To complete the exposition of the basic μ-ssRBM model, we present the free energy $f(v)$ of the visible vector.

$$\begin{aligned} f(v) &= -\log \sum_h \int \exp\{-E(v,s,h)\}\ ds \\ &= \frac{1}{2} v^T \Lambda v - \frac{1}{2} \sum_{i=1}^{N} \log\left(2\pi\alpha_i^{-1}\right) - \sum_{i=1}^{N} \log\left[1 + \exp\left\{-\frac{1}{2} v^T C_{v|h_i}^{-1} v + v^T W_i \mu_i + b_i\right\}\right] \end{aligned}$$

6.3 Positive Definite Parameterizations of the μ-ssRBM

Equation 6.6 reveals an important property of the μ-ssRBM model. The conditional $p(v \mid h)$ is only a well-defined Gaussian distribution if the covariance matrix $C_{v|h}$ is positive definite (PD). However, the covariance matrix is not parametrized to guarantee that this condition is met. If there exists a vector x such that $x^T C_{v|h} x \leq 0$, then the covariance matrix is not positive definite. In the original presentation of the ssRBM in Courville et al. (2010), the possibility of the non-positive-definiteness

of the conditional covariance of v given h was dealt with by limiting the support over the domain of v (i.e., $\mathbb{R}^D$) to a large but finite ball that encompasses all the training data. Such an approach is feasible but is aesthetically unsatisfying. A simple practical solution would be to mix a model with a ball constraint on $||v||$ with a very flat Gaussian that will catch any outliers outside the ball.

Here, in the context of the μ-ssRBM model, we turn to the question of how we can constrain the model parameters to guarantee that the model remains well-behaved (i.e., all conditionals are well-defined probability densities). The problem we face is ensuring that the covariance or equivalently the precision matrix of $p(v \mid h)$ is positive definite, i.e., we wish to satisfy the constraint:

$$x^T C_{v|h}^{-1} x > 0 \quad \forall x \neq \mathbf{0}. \tag{6.10}$$

To satisfy this constraint, we need to ensure that $\Lambda + \sum_{i=1}^{N} \Phi_i h_i$ is large enough to offset $\sum_{i=1}^{N} \alpha_i^{-1} W_i W_i^T h_i$. We consider two basic strategies: (1) define Λ to be large enough to offset a worst-case setting of the h; and (2) define the Φ_i to ensure that the contribution of each active h_i is itself PD.

6.3.1 Constraining Λ

One option to ensure that $C_{v|h}$ remains PD for all patterns of h activation is to constrain Λ to be large enough. In setting a constraint on Λ, we will ignore the contribution of the Φ_i terms (which leads to non-tightness of the constraint). Since the contribution of every $\alpha_i^{-1} W_i W_i^T h_i$ term is negative semi-definite, the worst case setting of the h would be to have $h_i = 1$ for all $i \in [1, N]$. This implies that Λ must be constrained such that:

$$x^T \left(\Lambda - \sum_{i=1}^{N} \alpha_i^{-1} W_i W_i^T \right) x > 0 \quad \forall x \neq \mathbf{0}. \tag{6.11}$$

If we take Λ to be a scalar matrix: $\Lambda = \lambda I$, then the problem of enforcing a PD precision matrix reduces to ensuring that λ is greater than the maximum eigenvalue

ρ of $\sum_{i=1}^{N} \alpha_i^{-1} W_i W_i^T$. In practice we can use the power iteration method to quickly estimate ρ, set an upper bound on the maximum eigenvalue, and then simply constrain $\lambda > \rho$ throughout training.

6.3.2 Constraining Φ

Another option to ensure that $C_{v|h}$ remains PD for all patterns of h activation is to constrain Φ_i to be large enough. Let W_{ij} be the jth element of the filter W_i (or equivalently, the ijth element of the weight matrix W) and let Φ_{ij} denote the ijth element of the diagonal Φ_i matrix. We want to choose Φ_i such that:

$$J(x, \Phi_i) = \sum_j x_j^2 \Phi_{ij} - (\sum_j w_{ij} x_j)^2 > 0 \quad \forall x \in \mathbb{R}^D, x \neq \mathbf{0}. \tag{6.12}$$

This condition will be satisfied for all x if we can satisfy it for the $x = u$ of norm 1 that minimizes $J(x, \Phi_i)$ (u is the eigenvector of the smallest eigenvalue of the matrix $\left(\Phi_i - \alpha_i^{-1} W_i W_i^T\right)$). To find u, we define the Lagrangian $L(x, \Phi_i) = J(x, \Phi_i) + \eta(1 - \sum_j x_j^2)$ to enforce the constraint $\sum_j x_j^2 = 1$. Setting $\frac{\partial L}{\partial x_j} = 0$ we recover the set of constraints:

$$\sum_j \frac{W_{ij}^2}{\Phi_{ij} - \eta} = 1, \tag{6.13}$$

$$\sum_j \frac{W_i^2 \Phi_{ij}}{(\Phi_{ij} - \eta)^2} > 1, \tag{6.14}$$

$$\eta > 0. \tag{6.15}$$

Consider the following general parametrization of Φ_i: $\Phi_{ij} = \eta + q\alpha^{-1} \frac{W_{ij}^2}{\beta_{ij}}$. This particular form is chosen so that our constraint set gives us $q = \sum_j \beta_{ij}$ and $\beta_{ij} > 0$. So with Φ_{ij} parametrized as $\Phi_{ij} = \zeta_{ij} + \alpha^{-1} \frac{W_{ij}^2}{\beta_{ij} / \sum_j \beta_{ij}}$ with $\zeta_{ij} > 0$, the covariance matrix of $p(v \mid h)$ is guaranteed to be PD. The parameter ζ_{ij} is an extra degree of freedom to Φ_{ij} to be estimated through maximum likelihood learning.

We are free to choose the parametrization of the β_{ij} provided $\beta_{ij} > 0$. For

example, with the choice $\beta_{ij} = W_{ij}^2$, Φ_i simplifies to

$$\Phi_{ij} = \zeta_{ij} + \alpha^{-1} \sum_j W_{ij}^2 I \tag{6.16}$$

where Φ_{ij} takes the form of a scalar matrix. Alternatively, we could choose $\beta_{ij} = 1$ with the result that

$$\Phi_{ij} = \zeta_{ij} + \alpha^{-1} D W_{ij}^2 \tag{6.17}$$

where the jth elements on the diagonal of Φ_i is scaled with W_{ij}^2.

While we are free to chose $\beta_{ij} > 0$ as we would like, the decision affects the inductive bias of the model. In the case of the Φ_{ij} parametrization given in Eqn. 6.16, the presence of the $\sum_i^N \Phi_i h_i$ as a scaling on the mean of the conditional $p(v \mid s,h)$ (Eqn. 6.4) implies that the activation of any h_i will have an effect on the scaling of the mean across the entire visible vector (or layer) irrespective of how localized is the corresponding filter W_i. Unsurprisingly, use of this parametrization tends to encourage both sparsely active h_i and W_i with relatively large receptive fields.

The Φ_i parametrization via Eqn. 6.17 has the property that the Φ_i receptive fields are steered in the direction of W_i. When W_i is near zero, Φ_i has little effect (unless it is mediated by ζ_i). This is an appealing property for modeling images or other data that give rise to sparse receptive fields W_i.

A third alternative that offers a compromise between these two parametrizations of Φ_i is obtained when $\beta_{ij} = W_{ij}^2 + \varepsilon$ and

$$\Phi_{ij} = \zeta_{ij} + \alpha^{-1} \frac{W_{ij}^2}{W_{ij}^2 + \varepsilon} \sum_j \left(W_{ij}^2 + \varepsilon \right). \tag{6.18}$$

In this case, for $W_{ij} \ll \varepsilon$ the Φ_i behaves similarly to that in Eqn. 6.17 narrowing the influence of Φ_i to only those dimensions with a significant W_{ij}^2. However, for $W_{ij} \gg \varepsilon$, Φ_i behaves like that in Eqn. 6.16.

6.3.3 Comparing strategies

The two general strategies to guarantee that the covariance matrix of $p(v \mid h)$ is positive definite are in some sense complementary. In the sparse operating regime of the μ-ssRBM (most h_i are inactive over most of the dataset), the $\mathbf{\Lambda}$ worst case assumption that $\forall i : h_i = 1$, becomes increasingly inaccurate and, as a result, the constraint on $\mathbf{\Lambda}$ becomes increasingly conservative. Therefore in the sparse regime, the $\mathbf{\Phi}$ constraints would seem more appropriate. On the other hand, in a highly non-sparse regime, the individual contributions of the $\mathbf{\Phi}$ to the global precision matrix can combine to form a more conservative PD precision matrix than would result from a constrained $\mathbf{\Lambda}$.

It is also possible to distribute responsibility for ensuring the constraint is satisfied jointly to $\mathbf{\Lambda}$ and $\mathbf{\Phi}$. This would constitute a mixed strategy, apportioning responsibility for compensating for a percentage of the negative definite $-\sum_{i=1}^{N} \alpha_i^{-1} W_i W_i^T h_i$ term to both $\mathbf{\Lambda}$ and $\mathbf{\Phi}$.

6.4 μ-ssRBM Learning and Inference

Learning and inference in the μ-ssRBM proceeds analogously to the original ssRBM and is rooted in the ability to efficiently draw samples from the model via Gibbs sampling. As with the original ssRBM, we seek a set of conditionals that will enable simple and efficient Gibbs sampling. Since sampling from the conditional $p(v \mid h)$ would involve the computationally prohibitive step of inverting a non-diagonal covariance matrix, we pursue a strategy of alternating sampling from the conditionals $P(h \mid v)$, $p(s \mid v,h)$ and $p(v \mid s,h)$. Each of these conditionals has the property that the distribution factors over the elements of the random vector, allowing us to efficiently draw samples from the model.

In training the μ-ssRBM, we use the stochastic maximum likelihood algorithm (SML, also known as persistent contrastive divergence) (Tieleman, 2008), where only one or a few Markov Chain (Gibbs) simulations are performed between each parameter update. These samples are then used to approximate the expectations

over the model distribution $p(v,s,h)$.

The data log likelihood gradient is

$$\frac{\partial}{\partial \theta_i}\left(\sum_{t=1}^{T}\log p(v_t)\right) = -\sum_{t=1}^{T}\left\langle \frac{\partial}{\partial \theta_i}E(v_t,s,h)\right\rangle_{p(s,h|v_t)} + T\left\langle \frac{\partial}{\partial \theta_i}E(v,s,h)\right\rangle_{p(v,s,h)}$$

This log likelihood gradient takes the form of a difference between two expectations, with the expectations over $p(s,h \mid v_t)$ in what is called the "clamped" condition, and the expectation over $p(v,s,h)$ in the so-called "unclamped" condition. As with the standard RBM, the expectations over $p(s,h \mid v_t)$ are amenable to analytic evaluation and so sampling is not necessary in the clamped condition.

6.5 Comparison to Previous Work

There is now a significant body of work on modelling natural images with RBM-based models. The closest connection to this work is to the original ss-RBM (Courville et al., 2010) which we recover by setting the μ-ssRMB parameters $\mu_i = 0$ and $\Phi_i = \mathbf{0}$ for all $i \in [1,N]$. A slightly less obvious limiting case of the μ-ssRMB is the Gaussian RBM (GRBM). Setting Φ_i to be proportional to α_i^{-1} (as discussed in Section 6.3) and taking $\alpha_i = \alpha \to \infty$, we define a Dirac in s about μ, In this limit, the conditionals $p(v \mid h)$ and $P(h_i = 1 \mid v)$ (Eqns. 6.6 and 6.7 respectively) are given by:

$$\lim_{\alpha\to\infty} p(v \mid h) = \mathcal{N}\left(v;\ \Lambda^{-1}\sum_{i=1}^{N} W_i\mu_i h_i,\ \Lambda^{-1}\right) \tag{6.19}$$

$$\lim_{\alpha\to\infty} P(h_i = 1 \mid v) = \sigma\left(v^T W_i\mu_i + b_i\right). \tag{6.20}$$

If we fix $\mu = 1$ and $\Lambda = I$, we recover the Gaussian RBM conditionals. Note that the connection between the μ-ssRBM and the Gaussian RBM is mediated entirely by the μ parameter, the original ssRBM has no such connection with the GRBM.

Beyond the ssRBM, the closest models to the μ-ssRBM are the mean and covariance RBM (mcRBM) (Ranzato and Hinton, 2010) and the mean Product of

t-distributions model (mPoT) (Ranzato et al., 2010b). Like the μ-ssRBM, both of these are energy-based models where the conditional distribution over the visible units conditioned on the hidden variables is a multivariate Gaussian with nonzero mean and a non-diagonal covariance matrix. However the μ-ssRBM differs from these models in the way the conditional means and covariance interact. In both the mcRBM and the mPoT model, the means are modelled by the introduction of additional GRBM hidden units, whereas in the μ-ssRBM, each hidden unit can potentially contribute to both the conditional mean and covariance. For the ith hidden unit, the contribution to each is controlled by the relative values of the μ_i and α_i. With large $|\mu_i|$ and small α_i, the unit predominantly contributes to the conditional mean. Conversely, with large α_i and small $|\mu_i|$, the unit predominantly contributes to the conditional covariance. The advantage of the μ-ssRBM approach is that we are able to save one hyper-parameter by letting maximum likelihood induction optimize the trade-off between mean and covariance modelling.

The mPoT and mcRBM also differ from the μ-ssRBM in how they parametrize the covariance over the visible units. While the μ-RBM uses the h activations to pinch the precision matrix along the direction specified by the corresponding weight vector, both the mcRBM and the mPoT models use their latent variable activations to maintain constraints, decreasing in value to allow variance in the direction of the corresponding weight vector.

Interestingly, the μ-ssRBM term involving Φ_i is very similar to the covariance term in the mcRBM energy function. The difference is that our restriction to a diagonal Φ_i significantly limits what we can model with it. However, this restricted structure allows us to sample efficiently from the model using Gibbs sampling which is not available to either the mcRBM or the mPoT model. In addition, the roles of these covariance terms are also quite different. In the mcRBM, the covariance term is associated with the model's feature vectors. In the μ-ssRBM, we use the Φ_i both to help constrain the model's conditionals to be well-defined, and in conjunction with W, to help maximize the likelihood of the training data.

Finally, the covariance structure of the μ-ssRBM conditional $p(v \mid h)$ (Eqn. 6.6)

is very similar to the product of probabilistic principal components analysis (PoP-PCA) model (Williams, 2001) with components corresponding to the ssRBM weight vectors associated with the active hidden units ($h_i = 1$).

6.6 Experiments

We demonstrate the utility of the μ-ssRBM on the CIFAR-10 dataset by classifying the images and by sampling from the model. In particular, our experiments are directed toward exploring the properties of the different elements of the model, including the roles of $\boldsymbol{\mu}$ and $\boldsymbol{\Phi}$ and the effects of the various PD constraints on $\boldsymbol{\Lambda}$ and $\boldsymbol{\Phi}$.

Our experiments are based on the CIFAR-10 image classification dataset consisting of 40 000 training images, 10 000 validation images, and 10 000 test images. The images are 32-by-32 pixel RGB images. Each image is labelled with one of ten object categories (aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) according to the most prominent object in the image.

6.6.1 Classification

We evaluate the μ-ssRBM as a feature-extraction algorithm by plugging it into the classification pipeline developed by Coates et al. (2010). In broad strokes, the μ-ssRBM is fit to (192-dimensional) 8x8 RGB image patches, and then applied convolutionally to the 32x32 images. The image patches (starting from pixels between 0 and 255) on which the μ-ssRBM was trained were centred, and then normalized by dividing by the square root of their variance plus a noise-cancelling constant (10). The normalized patches were whitened by ZCA (Hyvärinen and Oja, 2000) with a small positive constant (.1) added to all eigenvalues. The resulting patches (Figure 6.1, top left) are mostly grey with high spatial frequencies amplified, and lower spatial frequencies attenuated. Our models were trained from the 16 non-overlapping 8x8 patches from each of the first 10 000 training set images in CIFAR-10 (for a total of 160 000 training examples).

Models were trained for one hundred thousand minibatches of 100 patches. On an NVIDIA GTX 285 GPU this training took on the order of 15 minutes for most models. We used SML training (Tieleman, 2008). Classification was done with an ℓ_2-regularized SVM. The SVM was applied to the conditional mean value of latent spike (h) variables, extracted from every 8x8 image patch in the 32x32 CIFAR-10 image. Prior to classification, our conditional h values were spatially pooled into 9 regions, analogous to the 4 quadrants employed in Coates et al. (2010). For a model with N hidden units, the classifier operated on a feature vector of $9N$ elements.

Table 6.I lists the performance of several variants on the μ-ssRBM model. For this comparison all variants were trained with a small amount of sparsity aimed at maintaining 15% activity, and were configured with 256 hidden units. The lines labelled PD correspond to models that were constrained to have positive definite covariance of $p(v \mid h)$ while the lines labelled no PD are not. If $\mu = \mathbf{0}$ appears in a line the corresponding model was trained with $\mu = 0$ or equivalently with the μ terms removed from the energy function. The nomenclature for Φ is analogous. This implies for instance that the original ssRBM model would correspond to the no PD, $\mu = \mathbf{0}$, $\Phi = \mathbf{0}$ condition.

Table 6.I reveals that it is possible to constrain Φ to enforce that $C_{x|h}$ is PD and achieve classification results that match that of the original ssRBM. However, if we take the same μ-ssRBM form and loosen the PD constraint, the model can perform much better. Also of note is that both the μ and the Φ terms seem to contribute approximately equally to improving the classification accuracy.

Table 6.II situates the performance of the μ-ssRBM in the literature of results on CIFAR-10. The μ-ssRBM performs better than the most closely-related models - the GRBM, cRBM, and mcRBM. Recent work by Coates et al. (2010) has shown that a feature-extractor based on K-means actually out-performs these energy-based approaches to feature extraction on CIFAR-10, in the limit of very large hidden unit counts. Future work will look at more effective training strategies for energy models in this regime.

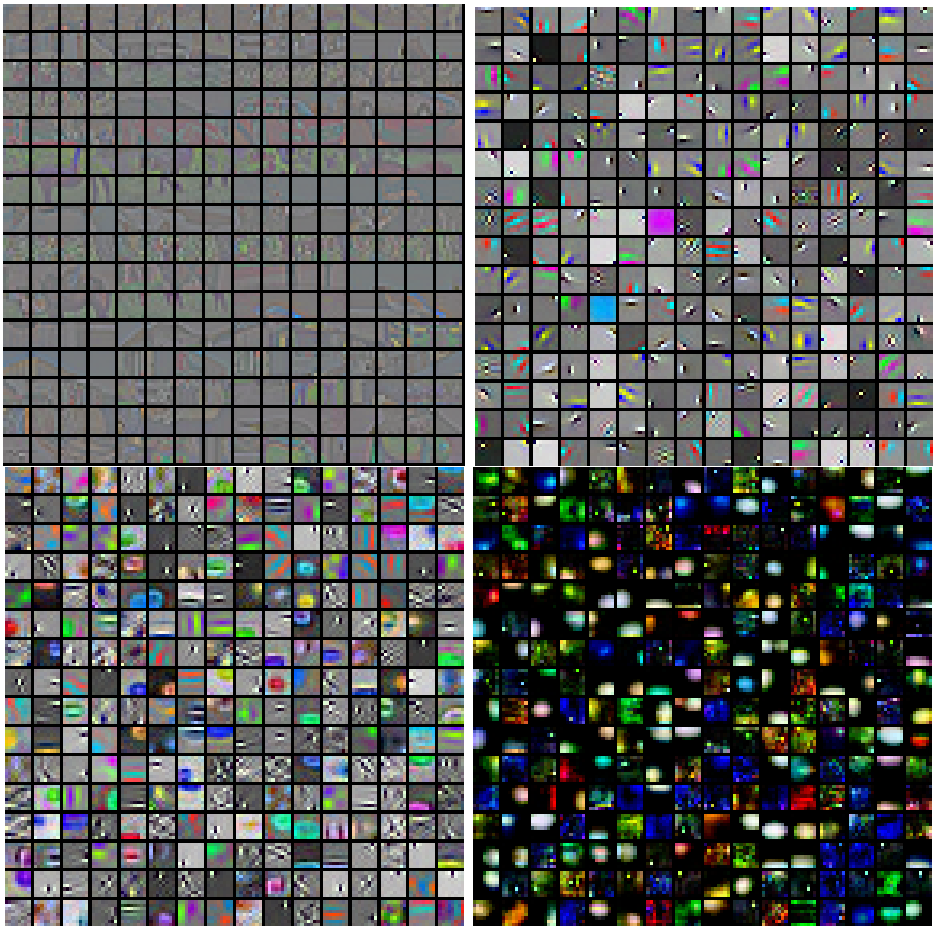

Figure 6.1: (Top left) ZCA-whitened data used for patch-wise training. (Top right) Filters W learnt when μ and Φ were fixed at zero. These filters produce edges similar to many other models, and neatly separate black-and-white edges from colour ones. Filters W (bottom left) and Φ (bottom right) learnt when μ and Φ are fit to the data. The combination of W and Φ gives individual units more flexibility, and gives rise to a richer variety of features that are better in classification.

Table 6.I: The performance of μ-ssRBM variants with 256 hidden units relative to one another in CIFAR-10 image classification. 95% confidence intervals are given for each score. Models labeled "no PD" were not constrained to be positive definite.

Model	Accuracy (%)
no PD, $\boldsymbol{\mu}$ free, $\boldsymbol{\Phi}$ free	73.1 $\pm$0.9
no PD, $\boldsymbol{\mu}$ free, $\boldsymbol{\Phi} = \mathbf{0}$	71.43 $\pm$0.9
no PD, $\boldsymbol{\mu} = \mathbf{0}$, $\boldsymbol{\Phi}$ free	71.19 $\pm$0.9
no PD, $\boldsymbol{\mu} = \mathbf{0}$, $\boldsymbol{\Phi} = \mathbf{0}$	68.92 $\pm$0.9
PD by Diag. W (Eqn. 6.17)	69.1$\pm$0.9
PD by $\boldsymbol{\Lambda}$ (Eqn. 6.11	68.3$\pm$0.9
PD by scal. mat. (Eqn. 6.16)	67.1$\pm$0.9

Table 6.II: The performance of μ-ssRBM variants relative to other models in the literature for CIFAR-10. 95% confidence intervals are given for each score assuming the official test set size of ten thousand. The k-means results are copied from Coates et al. (2010) The "conv. trained DBN" result is the convolutionally trained two-layer DBN with rectified linear units, described in Krizhevsky (2010) GRBM, cRBM, mcRBM results are copied from Ranzato and Hinton (2010)

Model	Accuracy (%)
k-means (4000 units)	79.6 $\pm$ 0.9
conv. trained DBN	78.9 $\pm$0.9
μ-ssRBM (4096 units)	76.7 $\pm$0.9
k-means (1200 units)	76.2 $\pm$ 0.9
μ-ssRBM (1024 units)	76.2 $\pm$0.9
k-means (800 units)	75.3 $\pm$ 0.9
μ-ssRBM (512 units)	74.1 $\pm$0.9
μ-ssRBM (256 units)	73.1 $\pm$0.9
k-means (400 units)	72.7 $\pm$ 0.9
k-means (200 units)	70.1 $\pm$ 0.9
mcRBM (225 factors)	68.2 $\pm$0.9
cRBM (900 factors)	64.7 $\pm$0.9
cRBM (225 factors)	63.6 $\pm$0.9
GRBM	59.7 $\pm$1.0

6.6.2 Sampling

To draw samples from the model, we trained it convolutionally, similarly to Krizhevsky (2010). Our convolutional implementation of the μ-ssRBM included 256 fully-connected units to capture global structure, and 128 hidden units per position for every position of an 8x8 RGB filter that fit within a padded CIFAR-10 image. The images were padded with a 3-pixel mirrored image border and attenuated, which is why the samples have a grey border. Filters W and Φ were shared across the image, though scalar-parameters μ_i, α_i, and hidden unit bias b_i were trained separately for each individual hidden unit (at each position). The model was trained as before by stochastic maximum likelihood, with the difference that the learning rate on the shared parameters W and Φ was reduced by a factor of 30. Figure 6.2 illustrates some samples drawn from the model, drawn by taking 500 Gibbs steps from training data. The 500 steps were enough for the samples to completely depart from the training data used to initialize the sampler. These samples exhibit global coherence, and sharp region boundaries, a range of colours, and natural-looking shading. Qualitatively, these samples are more varied and interesting than samples from similar energy-based models, such as those featured in Ranzato et al. (2010b).

6.7 Discussion

In this paper we have introduced the μ-ssRBM, a generalization of the ssRBM that includes extra terms in the energy function. One of the extra terms permits

Figure 6.2: Samples from a convolutionally trained μ-ssRBM exhibit global coherence, sharp region boundaries, a range of colours, and natural-looking shading.

the model to capture a non-zero mean in the Gaussian conditional $p(v \mid h)$, bringing the ssRBM framework in line with the recent work of Ranzato and Hinton (2010) and Ranzato et al. (2010b) which also modelled the conditional of the observed data given the latent variable value to be a general multivariate Gaussian with non-zero mean and full covariance. Unlike these other approaches, instantiating the slab vector s renders the μ-ssRBM amenable to efficient block Gibbs sampling.

The other functional term included in the μ-ssRBM energy function adds a positive definite diagonal contribution to the covariances associated with the Gaussian conditions over the observations. This term was used to define variants of the μ-ssRBM that were constrained to have well-defined conditional.

Still, our techniques for constraining the μ-ssRBM to have PD conditionals are based on loose worst-case scenarios, and potentially leave room for improvement. Our classification experiments indicate that the μ-ssRBM was able to use the extra capacity offered by the addition of these elements to the energy function to improve the classification accuracy. They also show that the addition of the PD constraint comes at the cost of classification performance.

CHAPTER 7

RANDOM HYPER-PARAMETER SELECTION

title	Random Search for Hyper-Parameter Optimization
author	James Bergstra and Yoshua Bengio
Publication	Submitted to the Journal of Machine Learning Research, March 2011.

This work compares grid search to other non-adaptive strategies for hyper-parameter optimization, such as random guessing and low-discrepancy sequences (e.g. Sobol, Halton) developed for Quasi-Monte-Carlo integration. Considering several datasets and two learning algorithms known for having many hyper-parameters (neural networks and deep belief networks) we find that grid search is the slowest and least-reliable method among those considered. When we view a learning algorithm as a function from a hyper-parameter vector to a generalization score, we find that these functions are typically dominated in any given neighbourhood by a small fraction of the hyper-parameters, and show that this kind of geometry is not well suited to lattice-based (grid) exploration. Our analysis casts some light on why recent "High Throughput" methods achieve somewhat surprising success – we suggest that they appear to search through a large number of hyper-parameters because most of those hyper-parameters matter relatively little. We anticipate that growing interest in large hierarchical models will place an increasing burden on techniques for hyper-parameter optimization. This work shows that random search is a natural baseline against which to judge progress in the development of sequential hyper-parameter optimization algorithms.

7.1 Introduction

Machine learning is often about finding a model $\theta^{(*)}$ in some set Θ that minimizes an expected loss $\mathscr{L}(x;\theta)$ over i.i.d. samples x from a natural (grand truth) distribution $\mathscr{G}_x$ (Bottou, 1998).

$$\theta^{(*)} = \text{argmin}_{\theta\in\Theta}\, \mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}(x;\theta)] \tag{7.1}$$

However, many sets Θ of interest (e.g. the set of all support vector machines, the set of all neural networks, the set of all classifiers implemented in a particular library) are difficult to search efficiently. A learning algorithm $\mathscr{A}$ can be seen as a function that maps datasets to models $\theta \in \Theta$, but often we have an efficient algorithm $\mathscr{A}$ (e.g. a gradient-based algorithm) that can search only some *part* of Θ. For example, neural networks for classification are typically optimized using gradient-based algorithms, but these gradient-based algorithms cannot optimize the (integer-valued) number of hidden units. When this occurs, it is conceptually convenient to separate the parameters θ describing a model into two kinds: *parameters* ϕ (which we can optimize efficiently) and *hyper-parameters* λ (which we cannot optimize efficiently). Where $\mathscr{A}$ has configuration variables of its own, we will include these into λ too.

When a learning algorithm $\mathscr{A}$ forces us to divide θ in this way, we have $\theta = (\phi,\lambda) \in (\Phi\times\Lambda)$. We can incorporate $\mathscr{A}$ into Equation 7.1 by splitting the argmin over θ into separate minimizations over Λ and Φ,

$$\begin{aligned}\lambda^{(*)} &= \underset{\lambda\in\Lambda}{\text{argmin}} \left(\min_{\phi\in\Phi} \; \mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}(x;\phi,\lambda)] \right) \\ &= \underset{\lambda\in\Lambda}{\text{argmin}} \; \mathbb{E}_{x\sim\mathscr{G}_x}[\mathscr{L}(x;\mathscr{A}(\lambda))]. \end{aligned} \tag{7.2}$$

By construction, we do not have efficient algorithms for performing the optimization implied by Equation 7.2. The (inefficient) technique for performing this optimization is called cross-validation, it is simply the method of trying out several

values $\boldsymbol{\lambda}^{(1)},...,\boldsymbol{\lambda}^{(S)}$ for $\boldsymbol{\lambda}$ on a *validation set* $\mathscr{X}^{(\text{valid})}$ of data $x \sim \mathscr{G}_x$ that is independent of any *training set* used internally by $\mathscr{A}$, and independent of the test set $\mathscr{X}^{(\text{test})}$ used later to evaluate $\boldsymbol{\lambda}^{(*)}$ (see Bishop, 1995, pp. 32-33). In practice we estimate $\boldsymbol{\lambda}^{(*)}$ using the procedure suggested by Equation 7.3.

$$\boldsymbol{\lambda}^{(*)} \approx \underset{\boldsymbol{\lambda} \in \{\boldsymbol{\lambda}^{(1)},...,\boldsymbol{\lambda}^{(S)}\}}{\text{argmin}} \; \text{mean}_{x \in \mathscr{X}^{(\text{valid})}} \; \mathscr{L}(x; \mathscr{A}(\boldsymbol{\lambda})). \tag{7.3}$$

$$= \underset{\boldsymbol{\lambda} \in \{\boldsymbol{\lambda}^{(1)},...,\boldsymbol{\lambda}^{(S)}\}}{\text{argmin}} \; \Psi(\boldsymbol{\lambda}) \tag{7.4}$$

This paper is about how to do cross-validation when the number (S) of hyper-parameter values in $\boldsymbol{\Lambda}$ becomes so large that the brute force approach suggested by Equation 7.3 becomes too computationally expensive. Large values of S are common because different hyper-parameters often vary independently of one another. The number of possible *joint values* for $\boldsymbol{\lambda}$ is the product of the number of possible values of each one. Grid search of K variables $l^{(1)}...l^{(K)}$ in $\boldsymbol{\Lambda}$ is the method of choosing sets of values $L^{(1)}...L^{(K)}$ for the variables, and trying each of the $S = \prod_k |L^{(k)}|$ elements that are composed by elements from those sets. This product over K sets makes grid search suffer from the so-called *curse of dimensionality*, because the number of joint values grows exponentially with the number of hyper-parameters (Bellman, 1961).

Cross-validation can be seen as an optimization algorithm of a function that maps from a hyper-parameter joint value to an expected loss. In this paper such a function will be called a *hyper-parameter response function*, and denoted by a $\boldsymbol{\Psi}$ (Equation 7.4). (This function is sometimes called the *response surface* in experiment design literature.) Brute force search is not the only way to optimize a such a function, but despite decades of research into global optimization (e.g. Kirkpatrick et al. (1983); Nelder and Mead (1965); Powell (1994); Weise (2009)) and the publishing of several hyper-parameter optimization algorithms (e.g. Czogiel et al. (2005); Hutter (2009); Nareyek (2003)), most machine learning researchers still prefer to carry out this optimization by hand, and by grid search (e.g. Hinton

(2010); Larochelle et al. (2007); LeCun et al. (1998b) as well as software packages such as libsvm (Chang and Lin, 2001) and scikits.learn [1]). We conjecture that there are two reasons.

- Manual optimization remains popular because researchers are interested in the behaviour of their algorithms in various conditions, and manual optimization provides that intuition as a side-effect of the optimization process.
- Grid search remains popular because it is simple, easily parallelized, and typically reliable because the function being optimized is too simple to require an adaptive (i.e. non-brute-force) optimization algorithm.

This paper argues that grid search (i.e., regular lattice-based, brute force search) should almost never be used. Instead, quasi-random or even pseudo-random experiment designs (random experiments) should be preferred. Random experiments are just as easily parallelized as grid search, just as simple to design, and more reliable. Adaptive search algorithms are more complicated to implement and are more difficult to parallelize. If a large compute cluster is available and random or quasi-random search is efficient enough, then these may always be the fastest algorithm (in terms of wall time) for hyper-parameter optimization.

Random search is efficient when the true hyper-parameter response function can be approximated accurately by a surrogate that looks at just a small number of hyper-parameters. It is not necessary that the identities of these hyper-parameters are known, only that the property holds. This property of having *low effective dimension* has been recognized in the literature of Quasi-Monte-Carlo (QMC) integration, and invoked to explain why QMC sometimes is much more efficient than worst-case analysis theory would predict. In the context of hyper-parameter search, the same notions of a low effective dimension and low discrepancy sets explains why grid search is so inefficient compared with experiments derived from either pseudo-random or quasi-random generators.

Like Drew and de Mello (2006), we draw the reader's attention to the low effective dimensionality of a function family of interest, as a means of optimizing it

1. `scikits.learn`: Machine Learning in Python (`http://scikit-learn.sourceforge.net`)

more efficiently. They present an algorithm that distinguishes between "important" and "unimportant" dimensions: QMC is used to choose points in the important dimensions, and unimportant dimensions are "padded" with thinner coverage and cheaper samples. Unlike them, we argue that for hyper-parameter selection (with a few hundred trials in 6-20 dimensions) there is little or no advantage to quasi-random generators compared with pseudo-random ones, and that the simplicity and i.i.d. nature of trials drawn from a pseudo-random distribution makes them preferable. The problem of distinguishing between important and unimportant hyper-parameter dimensions is a feature selection problem. We draw on this connection in our analysis in Section 7.5, and hope that this connection might inspire more efficient hyper-parameter optimization algorithms in future work.

This paper is organized as follows. Section 7.2 describes the datasets used in our experiments. Section 7.3 describes the bootstrap technique we use to estimate generalization in large cross-validation experiments. Section 7.4 describes the "random experiment efficiency curve" used to illustrate the results of our experiments. Section 7.5 repeats neural network experiments from Larochelle et al. (2007) with pseudo-random experiments, and shows (a) that random experiments are more efficient than grid ones and (b) how Gaussian Process regression characterizes the low effective dimensionality of the hyper-parameter response function in that setting. Section 7.5 also repeats deep belief network experiments from Larochelle et al. (2007) with pseudo-random ones to show that pseudo-random experiments are comparable to the heuristic hybrid manual and multi-resolution grid optimization strategy used in the original work; deep belief networks are not so hard to train after all. Section 7.6 describes how grid search becomes inefficient relative to pseudo-random and quasi-random experiments when a hyper-parameter response function has a low effective dimension, and shows in simulation that pseudo-random experiments are competitive with quasi-random ones in the regime of interest. The paper concludes in Section 7.7 that random experiments are generally better than grid ones for optimizing hyper-parameters.

7.2 Datasets

Following the work of Larochelle et al. (2007); Vincent et al. (2008), we use a variety of classification datasets that include many factors of variation. [2]

The **mnist basic** dataset is a subset of the well-known MNIST handwritten digit dataset (LeCun et al., 1998a). This dataset has 28x28 pixel grey-scale images of digits, each belonging to one of ten classes. We chose a different train/test/validation splitting in order to have faster experiments and see learning performance differences more clearly. We shuffled the original splits randomly, and used 10000 training examples, 2000 validation examples, and 50000 testing examples. These images are presented as white (1.0-valued) foreground digits against a black (0.0-valued) background.

The **mnist background images** dataset is a variation on **mnist basic** in which the white foreground digit has been composited on top of a 28x28 natural image patch. Technically this was done by taking the max of the original MNIST image and the patch. Natural image patches with very low pixel variance were rejected. As with **mnist basic** there are 10 classes, 10000 training examples, 2000 validation examples, and 50000 test examples.

The **mnist background random** dataset is a similar variation on **mnist basic** in which the white foreground digit has been composited on top of random uniform (0,1) pixel values. As with **mnist basic** there are 10 classes, 10000 training examples, 2000 validation examples, and 50000 test examples.

The **mnist rotated** dataset is a variation on **mnist basic** in which the images have been rotated by an amount chosen randomly between $\mathbf{0}$ and $\mathbf{2\pi}$ radians. This dataset included 10000 training examples, 2000 validation examples, 50000 test examples.

The **mnist rotated background images** dataset is a variation on **mnist rotated** in which the images have been rotated by an amount chosen randomly between $\mathbf{0}$ and $\mathbf{2\pi}$ radians, and then subsequently composited onto natural image

2. Datasets: `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/...`
`...Public/DeepVsShallowComparisonICML2007`

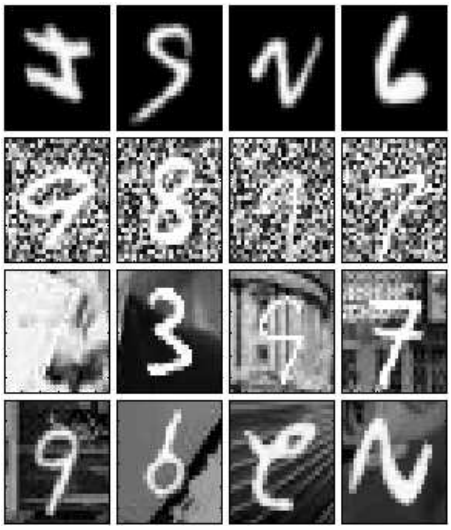

Figure 7.1: From top to bottom, samples from the **mnist rotated**, **mnist background random**, **mnist background images**, **mnist rotated background images** datasets. In all datasets the task is to identify the digit (0 - 9) and ignore the various distracting factors of variation.

patch backgrounds. This dataset included 10000 training examples, 2000 validation examples, 50000 test examples.

The **rectangles** dataset (Figure 7.2, top) is a simple synthetic dataset of outlines of rectangles. The images are 28x28, the outlines are white (1-valued) and the backgrounds were black (0-valued). The height and width of the rectangles were sampled uniformly, but when their difference was smaller than 3 pixels the samples were rejected. The top left corner of the rectangles was also sampled uniformly, with the constraint that the whole rectangle fits in the image. Each image is labelled as one of two classes: tall or wide. This task was easier than the MNIST digit classification, so we only used 1000 training examples, and 200 validation examples, but we still used 50000 testing examples.

The **rectangles images** dataset (Figure 7.2, bottom) is a variation on **rectangles** in which the foreground rectangles were filled with one natural image patch, and composited on top of a different background natural image patch. The process for sampling rectangle shapes was similar to the one used for **rectangles**, except a) the area covered by the rectangles was constrained to be between 25% and 75%

of the total image, b) the length and width of the rectangles were forced to be of at least 10, and c) their difference was forced to be of at least 5 pixels. This task was harder than **rectangles** so we used 10000 training examples, 2000 validation examples, and 50000 testing examples.

The **convex** dataset (Figure 7.3) is a binary image classification task. Each 28x28 image consists entirely of 1-valued and 0-valued pixels. If the 1-valued pixels form a convex region in image space, then the image is labelled as being convex otherwise it is labelled as non-convex. The convex sets consist of a single convex region with pixels of value 1.0. Candidate convex images were constructed by taking the intersection of a number of half-planes whose location and orientation were chosen uniformly at random. The number of intersecting half-planes was also sampled randomly according to a geometric distribution with parameter 0.195. A candidate convex image was rejected if there were less than 19 pixels in the convex region. Candidate non-convex images were constructed by taking the union of a random number of convex sets generated as above, but with the number of half-planes sampled from a geometric distribution with parameter 0.07 and with a minimum number of 10 pixels. The number of convex sets was sampled uniformly from 2 to 4. The candidate non-convex images were then tested by checking a convexity condition for every pair of pixels in the non-convex set. Those sets that failed the convexity test were added to the dataset. The parameters for generating the convex and non-convex sets were balanced to ensure that the conditional overall

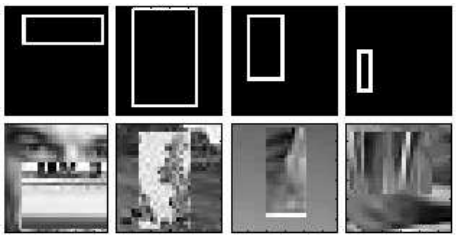

Figure 7.2: *Top:* Samples from the **rectangles** dataset. *Bottom:* Samples from the **rectangles images** dataset. In both datasets, the image is formed by overlaying a small rectangle on a background. The task in both datasets is to label the small rectangle as being either tall or wide.

pixel mean is the same for both classes.

7.3 Estimating Generalization

In large experiments, it often happens that there is a tie for the best validation set performance. This section describes our procedure for estimating test set accuracy, which takes into account any uncertainty in the choice of which trial is actually the best-performing one. To explain this procedure, we must distinguish between estimates of performance $\Psi^{(\text{valid})} = \Psi$ and $\Psi^{(\text{test})}$ based on the validation and test sets respectively:

$$\Psi^{(\text{valid})}(\lambda) = \text{mean}_{x \in \mathscr{X}^{(\text{valid})}} \; \mathscr{L}(x; \mathscr{A}(\lambda)), \tag{7.5}$$

$$\Psi^{(\text{test})}(\lambda) = \text{mean}_{x \in \mathscr{X}^{(\text{test})}} \; \mathscr{L}(x; \mathscr{A}(\lambda)). \tag{7.6}$$

Likewise, we must define the estimated variance $\mathbb{V}$ about these means on the validation and test sets:

$$\mathbb{V}^{(\text{valid})}(\lambda) = \frac{\Psi^{(\text{valid})}(\lambda)\left(1 - \Psi^{(\text{valid})}(\lambda)\right)}{|\mathscr{X}^{(\text{valid})}| - 1}, \text{ and} \tag{7.7}$$

$$\mathbb{V}^{(\text{test})}(\lambda) = \frac{\Psi^{(\text{test})}(\lambda)\left(1 - \Psi^{(\text{test})}(\lambda)\right)}{|\mathscr{X}^{(\text{test})}| - 1}. \tag{7.8}$$

In our experiments, where the $\mathscr{L}$ is a zero-one loss, this variance is based the Bernoulli variance. With other loss functions the estimator of variance will generally be different.

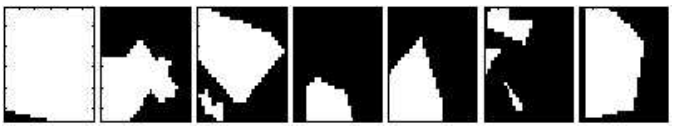

Figure 7.3: Samples from the **convex** dataset. The task is to identify whether the set of white pixels is convex. The first, fourth, and fifth images above are convex, the others are not.

The standard practice for evaluating a model found by cross-validation is to report $\Psi^{(\text{test})}(\lambda^{(s)})$ for the $\lambda^{(s)}$ that minimizes $\Psi^{(\text{valid})}(\lambda^{(s)})$. However, when different trials have nearly optimal validation means, then it is not clear which test score to report. To resolve the difficulty of choosing a winner, we report a weighted average of all the test set scores, in which each one is weighted by the probability that its particular $\lambda^{(s)}$ is in fact the best. In this view, the uncertainty arising from $\mathscr{X}^{(\text{valid})}$ being a finite sample of $\mathscr{G}_x$ makes the test-set score of the "best model" among $\lambda^{(1)},...,\lambda^{(S)}$ a random variable, z. This score z is distributed according to a Gaussian mixture model whose S components have means $\mu_s = \Psi^{(\text{test})}(\lambda^{(s)})$, variances $\sigma_s^2 = \mathbb{V}^{(\text{test})}(\lambda^{(s)})$, and weights w_s defined by

$$w_s = P\left(Z^{(s)} > Z^{(s')}, \ \ \forall s \neq s'\right), \text{ where} \tag{7.9}$$

$$Z^{(i)} \sim \mathscr{N}\left(\Psi^{(\text{valid})}(\lambda^{(i)}), \mathbb{V}^{(\text{valid})}(\lambda^{(i)})\right). \tag{7.10}$$

To summarize, the performance z of the best model in an experiment of S trials has mean μ_z and standard error σ_z^2,

$$\mu_z = \sum_{s=1}^{S} w_s \mu_s, \text{ and} \tag{7.11}$$

$$\sigma_z^2 = \sum_{s=1}^{S} w_s \left(\mu_s^2 + \sigma_s^2\right) - \mu_z^2. \tag{7.12}$$

It is simple and practical to estimate weights w_s by simulation. The procedure for doing so is to repeatedly draw hypothetical validation scores $Z^{(s)}$ from Normal distributions whose means are the $\Psi^{(\text{valid})}(\lambda^{(s)})$ and whose variances are the standard errors $\mathbb{V}^{(\text{valid})}(\lambda^{(s)})$, and to count how often each trial generates a winning score. Since the test scores of the best validation scores are typically relatively close, w_s need not be estimated very precisely and a few tens or hundreds of hypothetical draws suffice.

7.4 The Random Experiment Efficiency Curve

Figure 7.4 illustrates the results of a random experiment: an experiment of 256 trials training neural networks to classify the rectangles dataset. Since the trials of a random experiment are independently identically distribution (i.i.d.), a random search experiment involving S i.i.d. trials can also be interpreted as N independent experiments of s trials, as long as $sN \leq S$. This interpretation allows us to estimate statistics such as the minimum, maximum, median, and quantiles of any random experiment of size $s < S$.

There are two general trends in random experiment efficiency curves, such as the one in Figure 7.4: a sharp upward slope of the lower extremes as experiments grow, and a gentle downward slope of the upper extremes. The sharp upward slope occurs because when we take the maximum over larger subsets of the S trials, the worst-performing trials are more rarely the best in any subset. It is natural that larger experiments find trials with better scores. The shape of this curve indicates the frequency of good models under random search, and quantifies the relative volumes (in search space) of the various levels of performance. The gentle downward slope occurs because as we take the maximum over larger subsets of trials, we are less sure about which trial is actually the best. In large experiments we average together good validation trials with unexpectedly high test scores with other good validation trials with unexpectedly low test scores to arrive at a more accurate estimate.

Figure 7.4 characterizes the range of performance that is to be expected from experiments of various sizes, which is valuable information to anyone trying to reproduce these results. If just four random trials fails to find a score of 70%, then the problem is likely not in hyper-parameter selection. If Figure 7.4 had included just one upper outlying point it would indicate random good fortune on the experimenter's part that might be difficult for others to reproduce, and would anyway disappear in the average of Eqn. 7.11 after more trials.

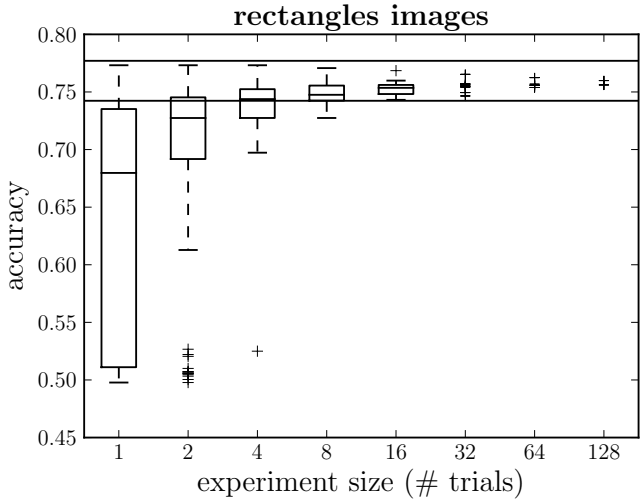

Figure 7.4: A random experiment efficiency curve. The trials of a random experiment are i.i.d, so an experiment of many trials (here, 256 trials optimizing a neural network to classify rectangles) can be seen equally as several independent smaller experiments. For example, at horizontal axis position 8, we consider our 256 trials to be 32 experiments of 8 trials each. The vertical axis shows the generalization error of the best trial(s) from experiments of a given size, as determined by Eqn. 7.11. When there is sufficiently many experiments of a given size (i.e., 10), the distribution of performance is illustrated by a box plot whose boxed section spans the lower and upper quartiles and includes a line at the median. The whiskers above and below each boxed section show the position of the most extreme data point within 1.5 times the inter-quartile range of the nearest quartile. Data points beyond the whiskers are plotted with '+' symbols. When there are not enough experiments to support a box plot, as occurs here for experiments of 32 trials or more, the best generalization score of each experiment is shown by a scatter plot. The two thin black lines across the top of the figure mark the upper and lower boundaries of a 95% confidence interval on the generalization of the best trial overall (Eqn. 7.12).

7.5 Relative Efficiency of Random Search

In this section we repeat several of the experiments of Larochelle et al. (2007) using purely random experiments, to investigate the shape of the various hyper-parameter response functions Ψ that arise. We begin in this section with a look at hyper-parameter optimization in neural networks, and then use automatic relevance determination in Gaussian Processes to characterize the low effective dimensionality of the various Ψ. Section 7.5.3 looks at hyper-parameter optimization in Deep Belief Networks (DBNs).

7.5.1 Case Study: Neural Networks

In Larochelle et al. (2007), the hyper-parameters of the neural network were optimized by search over a grid of trials. We describe the hyper-parameter configuration space of our neural network learning algorithm in terms of the distribution that we will use to randomly sample from that configuration space. The first hyper-parameter in our configuration is the type of data preprocessing: with equal probability, one of (a) none, (b) normalize (centre each feature dimension and divide by its standard deviation), or (c) PCA (after removing dimension-wise means, examples are projected onto principle components of the data whose norms have been divided by their eigenvalues). Subsequently to choosing PCA preprocessing, we must choose how many components to keep, we choose a fraction of variance to keep with a uniform distribution between 0.5 and 1.0. Some authors choose to discard a few leading eigenvectors of the PCA as well but we did not do this (Hyvärinen et al., 2001). There have been several suggestions for how the random weights of a neural network should be initialized (we will look unsupervised learning "pretraining" algorithms later in Section 7.5.3). We experimented with two heuristics: (a) a hyper-parameter multiplier on random uniform draws from $(-1, 1)$ divided by the square-root of the number of inputs (LeCun et al., 1998b), and (b) random normal values scaled by the square root of 6 over the square root of the number of inputs plus hidden units (Bengio and Glorot, 2010). The weights themselves were chosen

using one of three random seeds to the Mersenne Twister pseudo-random number generator. In the case of the first heuristic, we chose a multiplier uniformly from the range $(0.2, 2.0)$. The number of hidden units was drawn log-uniformly [3] from 18 to 1024. We selected either a sigmoidal or tanh nonlinearity with equal probability. The output weights from hidden units to prediction units were initialized to zero. The cost function was the mean error over minibatches of either 20 or 100 (with equal probability) examples at a time. The optimization algorithm was stochastic gradient descent with [initial] learning rate ε_0 drawn log-uniformly from 0.001 to 10.0. We offered the possibility of an annealed learning rate via a time point t_0 drawn log-uniformly from 300 to 30000. The effective learning rate ε_t after t minibatch iterations was

$$\varepsilon_t = \frac{t_0 \varepsilon_0}{\max(t, t_0)}. \tag{7.13}$$

We permitted a minimum of 100 and a maximum of 1000 iterations over the training data, stopping if ever, at iteration t, the best validation performance was observed before iteration $t/2$. With 50% probability, an ℓ_2 regularization penalty was applied, whose strength was drawn log-uniformly from 3.1×10^{-7} to 3.1×10^{-5}. This sampling process covers roughly the same domain with the same density as the grid used in Larochelle et al. (2007), except for the optional preprocessing steps. The grid optimization of Larochelle et al. (2007) did not consider normalizing or keeping only leading PCA dimensions of the inputs.

We formed experiments for each dataset by drawing $S = 256$ trials from this distribution. The results of these experiments are illustrated in Figures 7.5 and 7.6. Random sampling of trials is surprisingly effective in these settings. Figure 7.5 shows that even among the fraction of jobs (71/256) that used no preprocessing, the random search was able to do better than the grid search employed in Larochelle et al. (2007). If we look at the full set of 256 trials (Figure 7.6) that consider normalizing the input and PCA preprocessing, we see that these preprocessing strategies

3. We will use the phrase *drawn log-uniformly* from A to B for $0 < A < B$ to mean drawing uniformly in the log domain between $\log(A)$ and $\log(B)$, exponentiating to get a number between A and B, and then, for integer-valued parameters, rounding to the nearest integer.

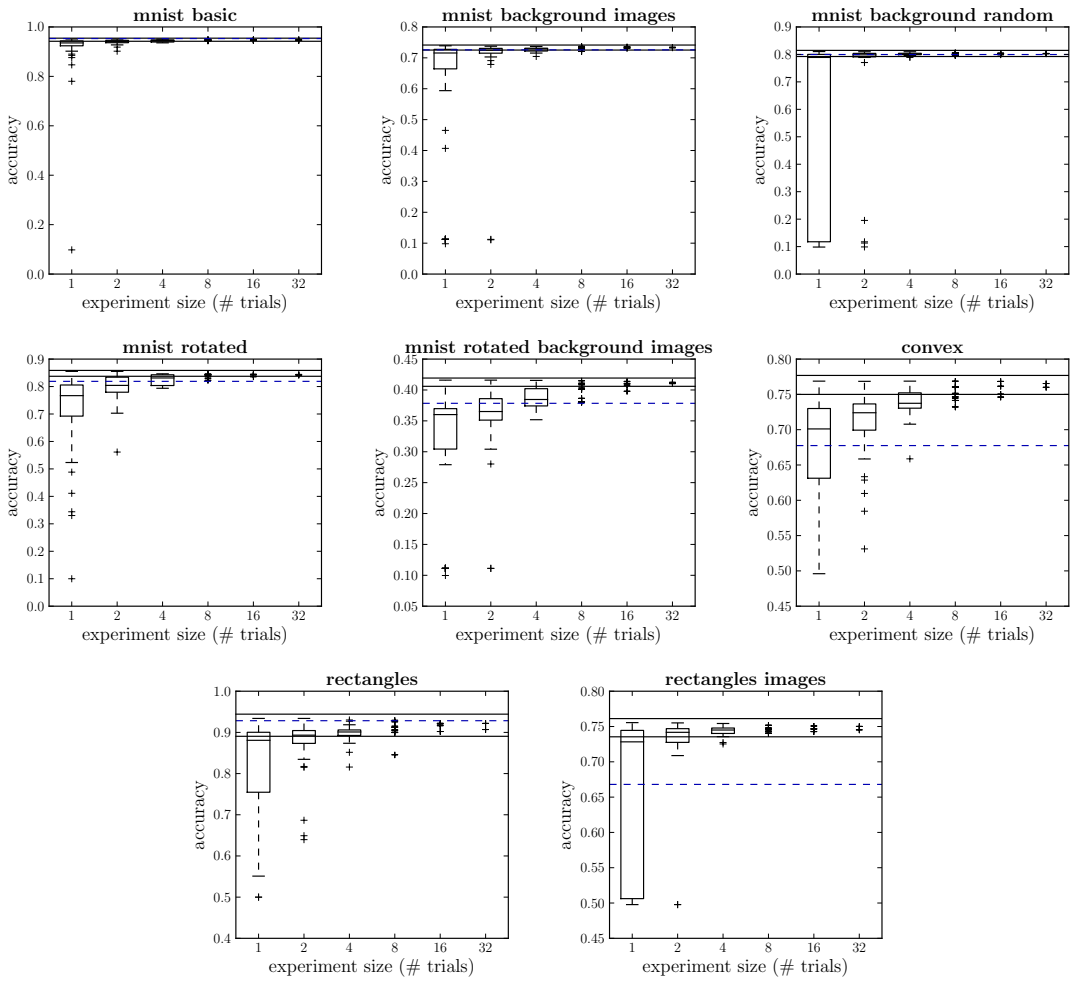

Figure 7.5: Neural network performance without preprocessing. Random experiment efficiency curves of a single-layer neural network for eight of the datasets used in Larochelle et al. (2007), looking only at trials with no preprocessing (7 hyper-parameters to optimize). The vertical axis is test set accuracy of the best model by cross-validation, the horizontal axis is the experiment size (the number of models compared in cross-validation). The dashed blue line represents grid search accuracy for neural network models based on a selection by grids averaging 100 trials (Larochelle et al., 2007). Random searches of 4 trials match or outperform grid searches.

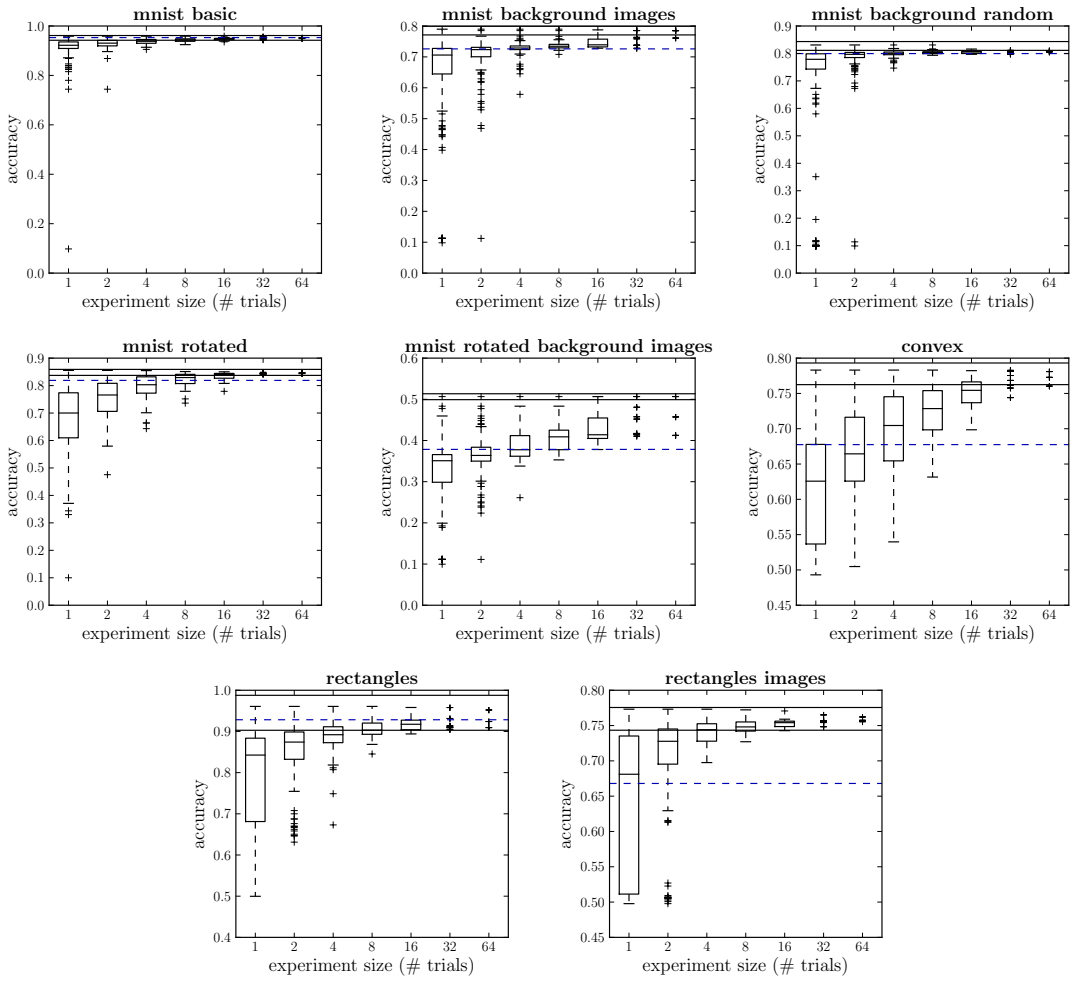

Figure 7.6: Neural network performance when standard preprocessing algorithms are considered (9 hyper-parameters). Dashed blue line represents grid search accuracy using (on average) 100 trials (Larochelle et al., 2007). Random searches of 32 trials consistently find better results than previous estimates neural network performance on these datasets.

improve performance. These random experiments of 256 trials are consistently better than grid search.

Note that the efficiency curves in Figures 7.5 and 7.6 reveal that different datasets give rise to functions Ψ with different shapes. The **mnist basic** results converge very rapidly toward what appears to be a global maximum. The fact that experiments of just 4 or 8 trials often have the same maximum as much larger experiments indicates that the region of Λ that gives rise to the best performance is approximately a quarter or an eighth of the entire configuration space. If indeed the random search has not missed a tiny region of significantly better preformance, then we can say that random search has solved this problem in 4 or 8 guesses. It is hard to imagine any optimization algorithm doing much better on a non-trivial 7-dimensional function. In contrast **mnist rotated background images** and **convex** experiments find that even with experiments of 16 or 32 random trials, there is considerable variation in the generalization of the reportedly best model. This reveals that the Ψ function is more peaked, with small regions of good performance that are still possible to find. For this sort of Ψ brute force techniques are unreliable. In all cases though, random experiments of 64 trials or more find consistently better results than grid search over an average of 100 trials.

7.5.2 The Low Effective Dimension of Ψ

The explanation for why random sampling of hyper-parameter configurations is so much more effective than grid-based sampling lies in the shape of the function Ψ. One simple way to characterize the shape of a high-dimensional function is to look at how much it varies in each dimension. Without a closed form for $\Psi(\boldsymbol{\lambda})$, we can still estimate the *relevance* of each hyper-parameter in $\boldsymbol{\lambda}$ to the value of $\Psi(\boldsymbol{\lambda})$. Gaussian process regression gives us the statistical machinery to look at Ψ in this way (Neal, 1998; Rasmussen and Williams, 2006).

We estimated the relevance of each hyper-parameter by fitting a Gaussian process (GP) with squared exponential kernels to predict $\Psi(\boldsymbol{\lambda})$ from $\boldsymbol{\lambda}$. The squared exponential kernel (also known as the Gaussian kernel) measures similarity between

two real-valued hyper-parameter values a and b by $e^{-\left(\frac{a-b}{l}\right)^2}$. The positive-valued l governs the sensitivity of the GP to changes in this hyper-parameter, or in other words, the *relevance* of the hyper-parameter to the GP prediction. In the case of the preprocessing hyper-parameter which assumed three categorical values, we used a dot-product kernel with an isotropic length scale matrix between 1-of-N feature vectors that encoded the category by the position of the 1. One wrinkle in our GP modelling was the fact that the hyper-parameter governing the fraction of PCA components to keep was irrelevant when the preprocessing was not by PCA. We dealt with this by imputing a value of 1.0 for the other preprocessing methods, because they did in effect retain all of the principle components. The kernels defined for each hyper-parameter were combined by multiplication.

Fitting a GP to Ψ means finding the relevance $(1/l)$ for each hyper-parameter that maximizes the likelihood of our data about Ψ. To compare relevance between hyper-parameters we divide each length scale by the range of the corresponding hyper-parameter. For hyper-parameters that were drawn log-uniformly (e.g. learning rate, number of hidden units), kernel calculations were based on the logarithm of the effective value.

Figure 7.7 shows the relevance of each component of h in modelling $\Psi(h)$. This figure reveals two important properties of Ψ for neural networks that explain why grid search performs so poorly relative to random experiments:

1. a small fraction of hyper-parameters matter for any one dataset,
2. but different hyper-parameters matter on different datasets.

These two properties taken together make grid search inefficient. To optimize Ψ with a grid, we must have enough values along each dimension to do a good job of optimizing that dimension when that dimension turns out to be relatively relevant. However, when (on other problems for example) that dimension turns out to be relatively irrelevant, then a grid experiment repeats nearly identical trials for each value along that dimension. This repetition makes the grid exponentially inefficient with respect to the number of relatively irrelevant dimensions.

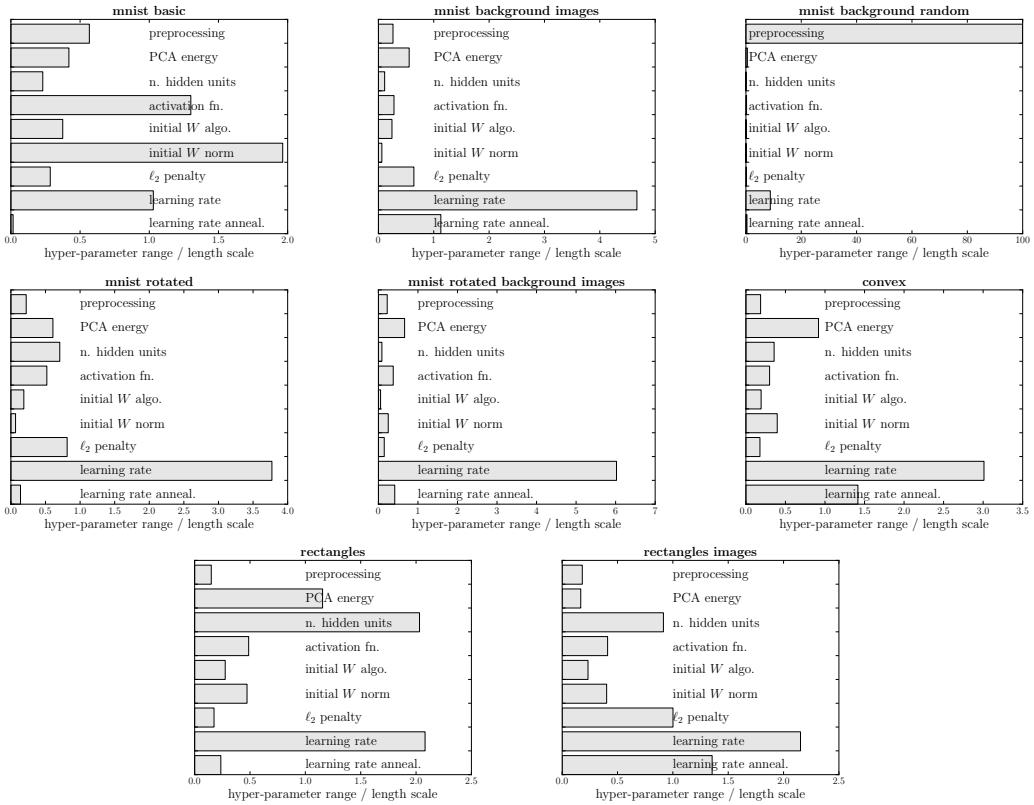

Figure 7.7: Automatic Relevance Determination (ARD) applied to hyper-parameters of neural network experiments. For each dataset, a small number of hyper-parameters dominate performance, but the relative importance of each hyper-parameter varies from one dataset to the next. Section 7.5.1 describes the nine hyper-parameters in each panel.

7.5.3 Case Study: Deep Belief Networks

To see how random search compares with grid-assisted hand-tuning of a model with many hyper-parameters, we look at the Deep Belief Network (DBN) model (Hinton et al., 2006). A DBN is a multi-layer graphical model with directed and undirected components. It is parameterized similarly to a multilayer neural network for classification, and it has been argued that *pretraining* a multilayer neural network by unsupervised learning as a DBN acts both to regularize the neural network toward better generalization, and to ease the optimization associated with *finetuning* the neural network for a classification task (Erhan et al., 2010).

A DBN classifier has many more hyper-parameters than a neural network. Firstly, there is the number of units and the parameters of random initialization for each layer. Secondly, there are hyper-parameters governing the unsupervised learning pretraining algorithm for each layer. Finally, there are hyper-parameters governing the global finetuning of the whole model for classification. For the details of how DBN models are trained (stacking restricted Boltzmann machines trained by contrastive divergence), the reader is referred to Larochelle et al. (2007), Hinton et al. (2006) or Bengio (2009). We evaluated random search by training 1-layer, 2-layer and 3-layer DBNs, sampling from the following distribution:

- We chose 1,2,or 3 layers with equal probability.
- For each layer, we choose:
  - a number of hidden units (log-uniformly between 128 and 4000),
  - a weight initialization heuristic that followed from a distribution (uniform or normal), a multiplier (uniformly between 0.2 and 2), a decision to divide by the fan-out (true or false),
  - a number of iterations of contrastive divergence to perform for pretraining (log-uniformly from 1 to 10000),
  - whether to treat the real-valued examples used for unsupervised pretraining as Bernoulli means (from which to draw binary-valued training samples) or as a samples themselves (even though they are not binary),

  - an initial learning rate for contrastive divergence (log-uniformly between 0.0001 and 1.0),
  - a time point at which to start annealing the contrastive divergence learning rate as in Equation 7.13 (log-uniformly from 10 to 10000).
- There was also the choice of how to preprocess the data. Either we used either the raw pixels or we removed some of the variance using a ZCA transform (in which examples are projected onto principle components, and then multiplied by the transpose of the principle components to place them back in the inputs space).
- If using ZCA preprocessing, we kept an amount of variance drawn uniformly from 0.5 to 1.0.
- We chose to seed our random number generator with one of 2, 3, or 4.
- We chose a learning rate for finetuning of the final classifier log-uniformly from 0.001 to 10.
- We chose an anneal start time for finetuning log-uniformly from 100 to 10000.
- We chose ℓ_2 regularization of the weight matrices at each layer during finetuning to be either 0 (with probability 0.5), or log-uniformly from 10^{-7} to 10^{-4}.

This hyper-parameter space includes 8 global hyper-parameters and 8 hyper-parameters for each layer, for a total of 32 hyper-parameters for 3-layer models.

A grid search is not practical for the 32-dimensional search problem of DBN model selection, because even just 2 possible values for each of 32 hyper-parameters would yield more trials than we could conduct ($2^{32} > 10^9$ trials and each can take hours). For many of the hyper-parameters, especially real valued ones, we would really like to try more than two values. The approach taken in Larochelle et al. (2007) was a combination of manual search, multi-resolution grid search and coordinate descent. The algorithm (including manual steps) is somewhat elaborate, but sensible, and we believe that it is representative of how model search is typically done in several research groups, if not the community at large. Larochelle et al. (2007) describe it as follows:

"The hyper-parameter search procedure we used alternates between fixing a neural network architecture and searching for good optimization hyper-parameters similarly to coordinate descent. More time would usually be spent on finding good optimization parameters, given some empirical evidence that we found indicating that the choice of the optimization hyper-parameters (mostly the learning rates) has much more influence on the obtained performance than the size of the network. We used the same procedure to find the hyper-parameters for DBN-1, which are the same as those of DBN-3 except the second hidden layer and third hidden layer sizes. We also allowed ourselves to test for much larger first-hidden layer sizes, in order to make the comparison between DBN-1 and DBN-3 fairer.

"We usually started by testing a relatively small architecture (between 500 and 700 units in the first and second hidden layer, and between 1000 and 2000 hidden units in the last layer). Given the results obtained on the validation set (compared to those of NNet for instance) after selecting appropriate optimization parameters, we would then consider growing the number of units in all layers simultaneously. The biggest networks we eventually tested had up to 3000, 4000 and 6000 hidden units in the first, second and third hidden layers respectively.

"As for the optimization hyper-parameters, we would proceed by first trying a few combinations of values for the stochastic gradient descent learning rate of the supervised and unsupervised phases (usually between 0.1 and 0.0001). We then refine the choice of tested values for these hyper-parameters. The first trials would simply give us a trend on the validation set error for these parameters (is a change in the hyper-parameter making things worse of better) and we would then consider that information in selecting appropriate additional trials. One could choose to use learning rate adaptation techniques (e.g. slowly decreasing the learning rate or using momentum) but we did not find these

techniques to be crucial.

There was large variation in the number of trials used in Larochelle et al. (2007) to optimize the DBN-3. One dataset (**mnist background images**) benefited from 102 grid trials, while another (**mnist background random**) only 13 because a good result was found more quickly. The average number of grid trials across datasets for the DBN-3 model was 41. In considering the number of trials per dataset, it is important to bear in mind that the experiments on different datasets were not performed independently. Rather, the experience drawn from earlier experiments affected later ones.

Random search versions of the DBN experiments from Larochelle et al. (2007) are shown in Figure 7.8. In this more challenging optimization problem random search is still effective, but not clearly superior as it was in the case of neural network optimization. Of the eight datasets used in our study, random search finds a better model than the manual search in one (**convex**), an equally good model in four (**mnist basic**, **mnist rotated**, **rectangles**, and **rectangles images**), and a poorer model in three (**mnist background images**, **mnist background random**, **mnist rotated background images**). In DBN optimization, we see that even experiments with larger numbers of trials (64 and larger) still show significant variability. This indicates that the regions of the search space with the best performance are small, and random search is not reliably finding them. To find the best hyper-parameter configurations reliably we must either use sequential optimization algorithms, or use knowledge about Ψ to make the search easier.

7.6 Low Effective Dimension

The setting in which a high-dimensional function can be approximated well by a function of a subset of its arguments has been studied in the context of numerical integration. A function with this property is said to have a *low effective dimension* (Caflisch et al., 1997). This property is invoked to explain why the variance of QMC estimators is often observed to decrease more quickly than worst-

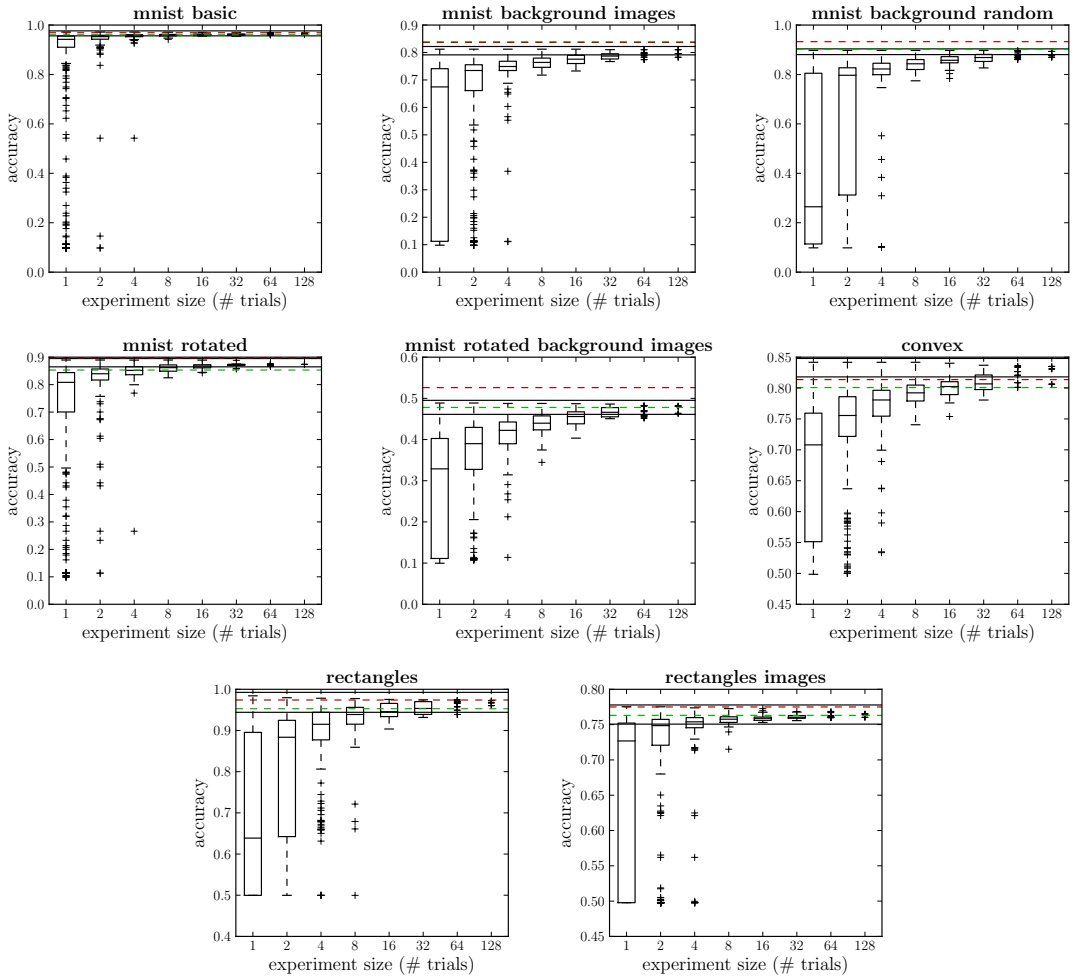

Figure 7.8: Deep Belief Network (DBN) performance according to random search over 32 hyper-parameters. Results of grid-assisted manual search using an average 41 trials are shown in green (1-layer DBN) and red (3-layer DBN). Random search finds a much better model on the **convex** dataset, but on other datasets the algorithms are tied, or else the grid-assisted manual search holds a slight advantage. The large variance in best-of-64 performance shows that the best-performing models occupy a relatively small region of the search space; sequential (rather than brute force) model selection algorithms will be necessary for effective optimization in such cases.

case analysis theory predicts. The worst-case analysis theory includes a term that is exponential in the dimensionality of the integration domain, and it has been argued that when a function has a low effective dimension, then it is the *effective* dimensionality that should be used to estimate the variance of the estimator.

In our setting we wish to optimize rather than integrate over the hyper-parameter configuration space, and that makes formal statements from the integration literature about low effective dimension inapplicable. Still, the basic principle of experiment efficiency is similar. In QMC integration, we minimize variance by designing a set of points with the property that they have *low discrepancy* with the uniform distribution. There are several definitions of low discrepancy, but they all try to capture the intuition that the points should be roughly equidistant from their neighbours, in order that there be no "clumps" or "holes" in the point set. This same notion of coverage is what we would like from our trial sets. The challenge is to design a set of trials so that no matter which dimensions turn out to be important, we will still have a relatively low-discrepancy set of points when we project all the trials onto the relevant subspace of the hyper-parameter domain $\mathbf{\Lambda}$. A grid of points appears to give even coverage, but a sub-space projection (especially one that is aligned with the axes of the grid) results in very clear clumps of points, which signal inefficient coverage of the low-dimensional space.

There are good choices beyond random (pseudo-random) point sets and grid sets. In the context of QMC integration, other low-discrepancy sequences have been introduced that are more uniform than pseudo-random draws and more robust than grids to projection. Sequences such as the Sobol, Halton, and Niederreiter [4] all have the property that low-dimensional projections of high-dimensional sequences are also good low-discrepancy sequences.

One might wonder that if random experiments are better than grid experiments, that experiments chosen based on low-discrepancy sequences might be better still.

4. These particular low-discrepancy sequences were chosen because they were implemented by the GNU Scientific Library (et al, 2009). More information about these sequences can be found as follows: Sobol (Antonov and Saleev, 1979), Halton (Halton, 1960), Niederrieter (Bratley et al., 1992),

Instead of running thousands of computationally demanding trials, we draw on the geometrical information about typical Ψ functions gleaned from our Gaussian Process analysis (Section 7.5.2) to address this question with a simulation.

The simulation search problem was to find a uniformly randomly placed multi-dimensional target interval, which occupies 1% of the volume of a unit hyper-cube. We looked at four variants of the search problem, in which the target was

1. a cube in a 3-dimensional space,
2. a box in a 3-dimensional space,
3. an equal-sided interval in a 5-dimensional space,
4. a hyper-rectangle in a 5-dimensional space.

The shape of the target rectangle in variants (2) and (4) was determined by sampling side lengths uniformly from the unit interval, and then scaling the rectangle to have a volume of 1%. This process gave the rectangles a shape that was often wide or tall - much longer along some axes than others. The position of the target was drawn uniformly among the positions totally inside the unit hyper-cube. In the case of tall or wide targets, the indicator function [of the target] has a lower effective dimension than the dimensionality of the overall space because the dimensions in which the target is elongated can be almost ignored.

The simulation experiment began with the generation of 100 search problems for each of the four search problem variants. Then for each experiment design method (grid, pseudo-random, Sobol, Halton, Niederreiter) we created experiments of 1, 2, 3, and up to 512 trials. There are many possible grid experiments of any size in multiple dimensions, especially for non-prime experiment sizes. We did not test every possible grid, instead we tested every grid with a monotonic resolution. For example, for experiments of size 16 in 5 dimensions, we tried the five grids with resolutions (1, 1, 1, 1, 16), (1, 1, 1, 2, 8), (1, 1, 2, 2, 4), (1, 1, 1, 4, 4), (1, 2, 2, 2, 2). Since the target intervals were generated in such a way that rectangles identical up to a permutation of side lengths have equal probability, grids with monotonic resolution are representative of all grids. The score of an experiment design method

for each experiment size was the fraction of the 100 targets that it found.

There are many possible random seeds and pseudo-random generators, so to characterize the performance of random search, we used the analytic expectation. The expected probability of finding the target is 1 minus the probability of missing the target with every single one of T trials in the experiment. If the volume of the target relative to the unit hypercube is $(v/V = 0.01)$ and there are T trials, then this probability of finding the target is

$$1 - (1 - \frac{v}{V})^T = 1 - .99^T \tag{7.14}$$

Figure 7.9 illustrates the efficiency of several low-discrepancy sequences at finding the multidimensional intervals, relative to the performance of all possible grids of each size and the expected performance of a pseudo-random experiment. There was no clear winner among these low-discrepancy sequences, but they were all much better at finding the high-dimensional non-square intervals than any of the basic grid experiments. In the 3-dimensional variants of the search problem, the experiments based on low-discrepancy sequences were more efficient than pseudo-random experiments when experiments were larger than 20 or 30 trials. However, in the 5-dimensional variants the advantage of low-discrepancy sequences over pseudo-random experiments disappeared. With just a few hundred trials, the coverage of low-discrepancy sequences is still so sparse that it is not readily distinguishable from pseudo-random points.

7.7 Conclusion

Grid search experiments are common in the literature of empirical machine learning, where they are used to optimize the hyper-parameters of learning algorithms. It is also common to perform multi-stage, multi-resolution grid experiments that are more or less automated, because a grid experiment with a fine-enough resolution for optimization would be prohibitively expensive. We have presented evidence that pseudo-random experiments are more efficient than grid experiments

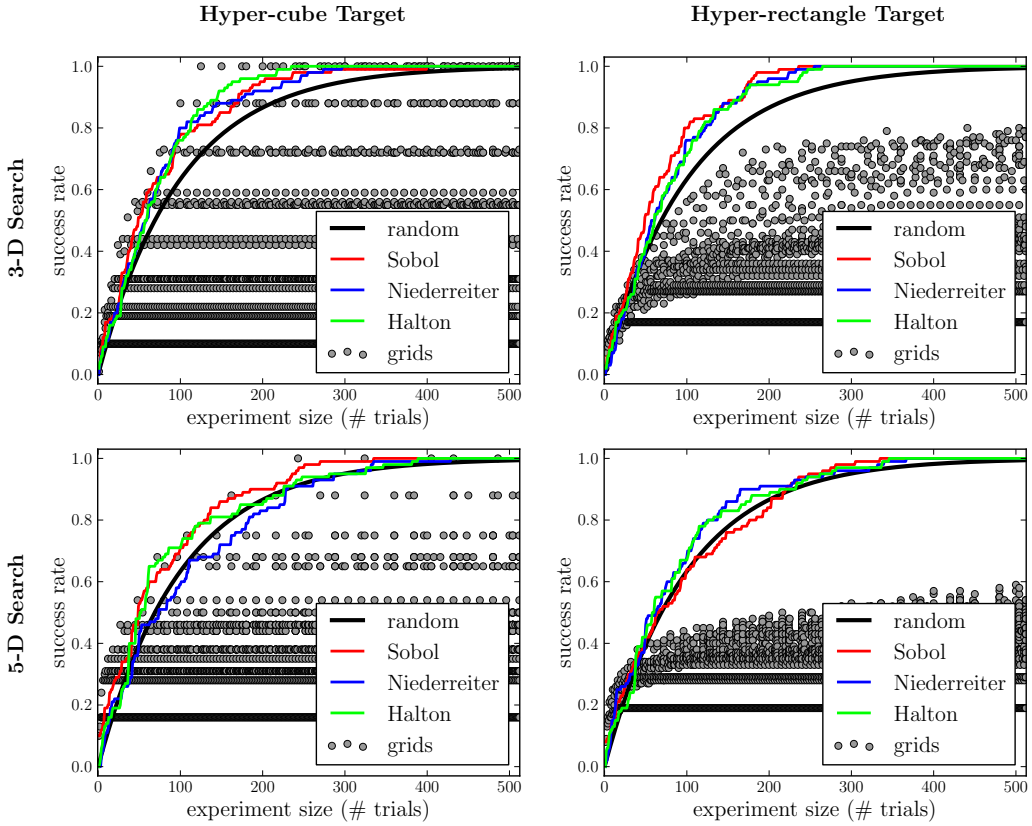

Figure 7.9: The efficiency in simulation of low-discrepancy sequences relative to grid and pseudo-random experiments. The simulation tested how reliably various experiment design methods locate a multidimensional interval occupying 1% of a unit hyper-cube. There is one grey circle in each sub-plot for every grid of every experiment size. Low-discrepancy sequences (tested: Sobol, Halton, Niederreiter) are slightly better than the expected performance (bold black) of pseudo-random guessing. Hyper-parameter search is most typically like the bottom-right scenario. Grid search experiments are inefficient for finding elongated regions in high dimensions (i.e., bottom-right). *Left:* All sides of the target interval have the same length in 3 and 5 dimensions. *Right:* The sides of the target interval have different lengths, finding the target requires higher search resolution in some dimensions than others.

for hyper-parameter selection for several kinds of learning algorithms on several datasets. Pseudo-random experiments are more efficient because hyper-parameter functions in practice include more- and less-important dimensions. Grid search experiments allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are critically important. We find that compared to the grid search experiments of Larochelle et al. (2007), a single random search is able to search the original experiment space *more efficiently than the heuristic, adaptive, multi-stage grid-based procedure*, and using fewer trials than the original experiments is able to find even better results both within the original configuration space and beyond. Critically, random search in these cases is a viable replacement for both the grid experiment and the researcher having to tune and refine the grid. Random search is a more reliable algorithm for hyper-parameter selection than grid search, just as easy to implement, just as easy to parallelize, and can be expected to find better models with fewer jobs.

Random experiments are also easier to carry out than grid experiments for practical reasons related to the statistical independence of every trial.

- The experiment can be stopped any time and the trials form a complete experiment.
- If extra computers become available, new trials can be added to an experiment without having to adjust the grid and commit to a much larger experiment.
- Every trial can be carried out asynchronously.
- If the computer carrying out a trial fails for any reason, its trial can be either abandoned or restarted without jeopardizing the experiment.

Random search is not incompatible with a controlled experiment. To investigate the effect of one hyper-parameter of interest X, we recommend random search (instead of grid search) for optimizing over other hyper-parameters. Choose one set of random values for these remaining hyper-parameters and use that same set for each value of X.

Random experiments with large numbers of trials also bring attention to the question of how to measure test error of an experiment when many trials have some

claim to being best. When using a relatively small validation set, the uncertainty involved in selecting the best model by cross-validation can be larger than the uncertainty in measuring the test set performance of any one model. It is important to take both of these sources of uncertainty into account when reporting the uncertainty around the best model found by a model search algorithm. This technique is useful to all of experiments (including both random and grid) in which multiple trials achieve approximately the best performance.

Low-discrepancy sequences developed for QMC integration are also good alternatives to grid-based experiments. In low dimensions (e.g. 1-4) our simulated results suggest that they can hold some advantage over pseudo-random experiments, but that in higher numbers of dimensions this advantage disappears. At the same time, the trials of a low-discrepancy experiment are not i.i.d. and that makes it more challenging or impossible (depending on the nature of the sequence) to subdivide one low-discrepancy experiment into several smaller low-discrepancy experiments. It is not generally possible to estimate performance statistics in the random efficiency curves from the results of an experiment based on a low-discrepancy sequence.

Finally, the hyper-parameter optimization strategies considered here are non-adaptive; that is to say they do not vary the course of the experiment by considering any results that may already be available. Future work should consider sequential, adaptive search/optimization algorithms in settings where many hyper-parameters must be optimized jointly and the effective dimensionality is high. Current work in that direction is promising (Hutter, 2009; Hutter et al., 2011; Srinivasan and Ramakrishnan, 2011). We hope that future work in that direction will consider random search of the form studied here as a baseline for performance, rather than grid search.

CHAPTER 8

THEANO

Title	Theano: A CPU and GPU Math Compiler in Python
Author	James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio
Publication	Proceedings of the 9^{th} Python in Science Conference (SCIPY 2010)

Theano is a compiler for mathematical expressions in Python that combines the convenience of NumPy's syntax with the speed of optimized native machine language. The user composes mathematical expressions in a high-level description that mimics NumPy's syntax and semantics, while being statically typed and functional (as opposed to imperative). These expressions allow Theano to provide symbolic differentiation. Before performing computation, Theano optimizes the choice of expressions, translates them into C++ (or CUDA for GPU), compiles them into dynamically loaded Python modules, all automatically. Common machine learning algorithms implemented with Theano are from 1.6× to 7.5× faster than competitive alternatives (including those implemented with C/C++, NumPy, and MATLAB) when compiled for the CPU and between 6.5× and 44× faster when compiled for the GPU. This paper illustrates how to use Theano, outlines the scope of the compiler, shows benchmarking results on both CPU and GPU processors, and explains its overall design.

8.1 Introduction

Python is a powerful and flexible language for describing large-scale mathematical calculations, but the Python interpreter is in many cases a poor engine for executing them. One reason is that Python uses full-fledged Python objects on the heap to represent simple numeric scalars. To reduce the overhead in numeric calculations, it is important to use array types such as NumPy's `ndarray` so that single Python objects on the heap can stand for multidimensional arrays of numeric scalars, each stored efficiently in the host processor's native format.

NumPy (Oliphant, 2007) provides an N-dimensional array data type, and many functions for indexing, reshaping, and performing elementary computations (`exp`, `log`, `sin`, etc.) on entire arrays at once. These functions are implemented in C for use within Python programs. However, the composition of many such NumPy functions can be unnecessarily slow when each call is dominated by the cost of transferring memory rather than the cost of performing calculations (Alted, 2010). The numexpr library [1] goes one step further by providing a loop fusion optimization that can glue several element-wise computations together. Unfortunately, numexpr requires an unusual syntax (the expression must be encoded as a string within the code), and at the time of this writing, numexpr is limited to optimizing element-wise computations. Cython (Behnel et al., 2009) and scipy.weave [2] address Python's performance issue by offering a simple way to hand-write crucial segments of code in C (or a dialect of Python which can be easily compiled to C, in Cython's case). While this approach can yield significant speed gains, it is labour-intensive: if the bottleneck of a program is a large mathematical expression comprising hundreds of elementary operations, manual program optimization can be time-consuming and error-prone, making an automated approach to performance optimization highly desirable.

Theano, on the other hand, works on a symbolic representation of mathemat-

1. numexpr: `http://code.google.com/p/numexpr/`
2. SciPy Weave module (`http://docs.scipy.org/doc/scipy/reference/tutorial/weave.html`)

ical expressions, provided by the user in a NumPy-like syntax. Access to the full computational graph of an expression opens the door to advanced features such as symbolic differentiation of complex expressions, but more importantly allows Theano to perform local graph transformations that can correct many unnecessary, slow or numerically unstable expression patterns. Once optimized, the same graph can be used to generate CPU as well as GPU implementations (the latter using CUDA) without requiring changes to user code.

Theano is similar to SymPy [3] in that both libraries manipulate symbolic mathematical graphs, but the two projects have a distinctly different focus. While SymPy implements a richer set of mathematical operations of the kind expected in a modern computer algebra system, Theano focuses on fast, efficient evaluation of primarily array-valued expressions.

Theano is free open source software, licensed under the New (3-clause) BSD license. It depends upon NumPy, and can optionally use SciPy. Theano includes many custom C and CUDA code generators which are able to specialize for particular types, sizes, and shapes of inputs; leveraging these code generators requires gcc (CPU) and nvcc (GPU) compilers, respectively. Theano can be extended with custom graph expressions, which can leverage `scipy.weave`, PyCUDA, Cython, and other numerical libraries and compilation technologies at the user's discretion. Theano has been actively and continuously developed and used since January 2008. It has been used in the preparation of numerous scientific papers and as a teaching platform for machine learning in graduate courses at l'Université de Montéal. Documentation and installation instructions can be found on Theano's website. [4] All Theano users should subscribe to the (low traffic) announce mailing list [5]. There are medium traffic mailing lists for developer [6] discussion and user support [7].

This paper is divided as follows: Section 8.2 shows how Theano can be used to

3. SymPy: Python Library for Symbolic Mathematics. (`http://www.sympy.org/`)
4. website: `http://www.deeplearning.net/software/theano`
5. Announcements: `http://groups.google.com/group/theano-announce`
6. Developer discussion: http://groups.google.com/group/theano-dev
7. User support: `http://groups.google.com/group/theano-users`

solve a simple problem in statistical prediction. Section 8.3 presents some results of performance benchmarking on problems related to machine learning and expression evaluation. Section 8.4 gives an overview of the design of Theano and the sort of computations to which it is suited. Section 8.5 provides a brief introduction to the compilation pipeline. Section 8.6 outlines current limitations of our implementation and planned additions to Theano.

8.2 Case Study: Logistic Regression

To get a sense of how Theano feels from a user's perspective, we will look at how to solve a binary logistic regression problem. Binary logistic regression is a classification model parameterized by a weight matrix W and bias vector b. The model estimates the probability $P(Y = 1|x)$ (which we will denote with shorthand p) that the input x belongs to class $y = 1$ as:

$$P(Y = 1|x^{(i)}) = p^{(i)} = \frac{e^{Wx^{(i)}+b}}{1 + e^{Wx^{(i)}+b}} \tag{8.1}$$

The goal is to optimize the log probability of N training examples $(x^{(i)}, y^{(i)})$ with respect to W and b (where $0 < i \leq N$). To maximize the log likelihood we will instead minimize the (average) negative log likelihood:

$$\mathscr{L}^{(\mathrm{NLL})}(W, b) = -\frac{1}{N} \sum_i y^{(i)} \log p^{(i)} + (1 - y^{(i)}) \log(1 - p^{(i)}) \tag{8.2}$$

To make it a bit more interesting, we can also include an ℓ_2 penalty on W, giving a cost function $\mathscr{L}(W, b)$ defined as:

$$\mathscr{L}(W, b) = \mathscr{L}^{(\mathrm{NLL})}(W, b) + 0.01 \sum_i \sum_j w_{ij}^2 \tag{8.3}$$

In this example, tuning parameters W and b will be done through stochastic gradient descent (SGD) on $\mathscr{L}(W, b)$. Stochastic gradient descent is a method for minimizing a differentiable loss function which is the expectation of some per-

example loss over a set of training examples. SGD estimates this expectation with an average over one or several examples and performs a step in the approximate direction of steepest descent. Though more sophisticated algorithms for numerical optimization exist, in particular for smooth convex functions such as $\mathscr{L}(W,b)$, stochastic gradient descent is a competitive method when the number of training examples is large or when training examples arrive in a continuous stream (Bottou, 1998).

The SGD algorithm repeatedly updates W as follows:

$$W \leftarrow W - \varepsilon \frac{1}{N'} \sum_i \left. \frac{\partial \mathscr{L}(W,b)}{\partial W} \right|_{x=x^{(i)}, y=y^{(i)}} \tag{8.4}$$

where ε is the learning rate and N' is the number of examples with which we will approximate the gradient. The update on b is likewise

$$b \leftarrow b - \varepsilon \frac{1}{N'} \sum_i \left. \frac{\partial \mathscr{L}(W,b)}{\partial b} \right|_{x=x^{(i)}, y=y^{(i)}}. \tag{8.5}$$

Implementing this minimization procedure in Theano involves the following four conceptual steps:

1. declaring symbolic variables,
2. using these variables to build a symbolic expression graph,
3. compiling Theano functions, and
4. calling said functions to perform numerical computations.

The code listings in Figures 8.1 - 8.4 illustrate these steps with a working program that fits a logistic regression model to random data.

The code in Figure 8.1 declares four symbolic variables `x`, `y` `w`, and `b` to represent the data and parameters of the model. Each tensor variable is strictly typed to include its data type, its number of dimensions, and the dimensions along which it may broadcast (like NumPy's broadcasting) in element-wise expressions. The variable `x` is a matrix of the default data type (`float64`), and `y` is a vector of type `long` (or `int64`). Each row `x[i]` will store an example $x^{(i)}$, and each element `y[i]`

```
 1: import numpy
 2: import theano.tensor as T
 3: from theano import shared, function
 4:
 5: x = T.matrix()
 6: y = T.lvector()
 7: w = shared(numpy.random.randn(100))
 8: b = shared(numpy.zeros(()))
 9: print "Initial model:"
10: print w.get_value(), b.get_value()
```

Figure 8.1: Logistic regression, part 1: declaring variables.

will store the corresponding label $y^{(i)}$. The number of examples to use at once represents a trade-off between computational and statistical efficiency.

The `shared()` function creates *shared variables* for W and b and assigns them initial values. Shared variables behave much like other Theano variables, with the exception that they also have a persistent value. A shared variable's value is maintained throughout the execution of the program and can be accessed with `.get_value()` and `.set_value()`, as shown in line 10.

```
11: p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
12: xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
13: cost = xent.mean() + 0.01*(w**2).sum()
14: gw,gb = T.grad(cost, [w,b])
15: prediction = p_1 > 0.5
```

Figure 8.2: Logistic regression, part 2: the computation graph.

The code in Figure 8.2 specifies the computational graph required to perform stochastic gradient descent on the parameters of our cost function. Since Theano's interface shares much in common with that of NumPy, lines 11-15 should be self-explanatory for anyone familiar with that module. On line 11, we start by defining $P(Y = 1|x^{(i)}) = 1$ as the symbolic variable `p_1`. Notice that the matrix multiplication and element-wise exponential functions are simply called via the `T.dot` and `T.exp` functions, analogous to `numpy.dot` and `numpy.exp`. `xent` defines the cross-entropy loss function, which is then combined with the ℓ_2 penalty on line 13, to form the cost function of Eqn. 8.3 and denoted by `cost`.

Line 14 is crucial to our implementation of SGD, as it performs symbolic differentiation of the scalar-valued `cost` variable with respect to variables `w` and `b`. `T.grad` operates by iterating backwards over the expression graph, applying the

chain rule of differentiation and building symbolic expressions for the gradients on `w` and `b`. As such, `gw` and `gb` are also symbolic Theano variables, representing $\partial\mathscr{L}/\partial W$ and $\partial\mathscr{L}/\partial b$ respectively. Finally, line 15 defines the actual prediction (the `prediction` variable) of the logistic regression by thresholding $P(Y = 1|x^{(i)})$.

```
16: predict = function(inputs=[x],
17:                    outputs=prediction)
18: train = function(
19:             inputs=[x,y],
20:             outputs=[prediction, xent],
21:             updates={w:w-0.1*gw, b:b-0.1*gb})
```

Figure 8.3: Logistic regression, part 3: compilation.

The code of Figure 8.3 creates the two functions required to train and test our logistic regression model. Theano functions are callable objects that compute zero or more *outputs* from values given for one or more symbolic *inputs*. For example, the `predict` function computes and returns the value of `prediction` for a given value of `x`. Parameters `w` and `b` are passed implicitly - all shared variables are available as inputs to all functions as a convenience to the user.

Line 18 (Figure 8.3) which creates the `train` function highlights two other important features of Theano functions: the potential for multiple outputs and updates. In our example, `train` computes both the prediction (`prediction`) of the classifier as well as the cross-entropy error function (`xent`). Computing both outputs together is computationally efficient since it allows for the reuse of intermediate computations, such as `dot(x,w)`. The optional `updates` parameter enables functions to have side-effects on shared variables. The updates argument is a dictionary which specifies how shared variables should be updated after all other computation for the function takes place, just before the function returns. In our example, calling the `train` function will update the parameters `w` and `b` with new values as per the SGD algorithm.

Our example concludes (Figure 8.4) by using the functions `train` and `predict` to fit the logistic regression model. Our data `D` in this example is just four random vectors and labels. Repeatedly calling the `train` function (lines 27-28) fits our parameters to the data. Note that calling a Theano function is no different

```
22: N = 4
23: feats = 100
24: D = (numpy.random.randn(N, feats),
25:      numpy.random.randint(size=N,low=0, high=2))
26: training_steps = 10
27: for i in range(training_steps):
28:     pred, err = train(D[0], D[1])
29: print "Final model:",
30: print w.get_value(), b.get_value()
31: print "target values for D", D[1]
32: print "prediction on D", predict(D[0])
```

Figure 8.4: Logistic regression, part 4: computation.

than calling a standard Python function: the graph transformations, optimizations, compilation and calling of efficient C-functions (whether targeted for the CPU or GPU) have all been done under the hood. The arguments and return values of these functions are NumPy `ndarray` objects that interoperate normally with other scientific Python libraries and tools.

8.3 Benchmarking Results

Theano was developed to simplify the implementation of complex high-performance machine learning algorithms. This section presents performance in two processor-intensive tasks from that domain: training a Multi-Layer Perceptron (MLP) and training a convolutional network. We chose these architectures because of their popularity in the machine learning community and their different computational demands. Large matrix-matrix multiplications dominate in the MLP example and two-dimensional image convolutions with small kernels are the major bottleneck in a convolutional network. More information about these models and their associated learning algorithms is available from the Deep Learning Tutorials.[8] The implementations used in these benchmarks are available online.[9]

CPU timing was carried out on an Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz with 2 GB of RAM. All implementations were linked against the BLAS implemented in the Intel Math Kernel Library, version 10.2.4.032 and allowed to use only one thread. GPU timing was done on a GeForce GTX 285. CPU computations

8. Deep Learning Tutorials: `http://deeplearning.net/tutorial/`
9. Benchmarking code: `http://github.com/pascanur/DeepLearningBenchmarks`

were done at double-precision, whereas GPU computations were done at single-precision.

Our first benchmark involves training a single layer MLP by stochastic gradient descent. Each implementation repeatedly carried out the following steps: (1) multiply 60 784-element input vectors by a 784×500 weight matrix, (2) apply an element-wise hyperbolic tangent operator (tanh) to the result, (3) multiply the result of the tanh operation by a 500×10 matrix, (4) classify the result using a multi-class generalization of logistic regression, (5) compute the gradient by performing similar calculations but in reverse, and finally (6) add the gradients to the parameters. This program stresses element-wise computations and the use of BLAS routines.

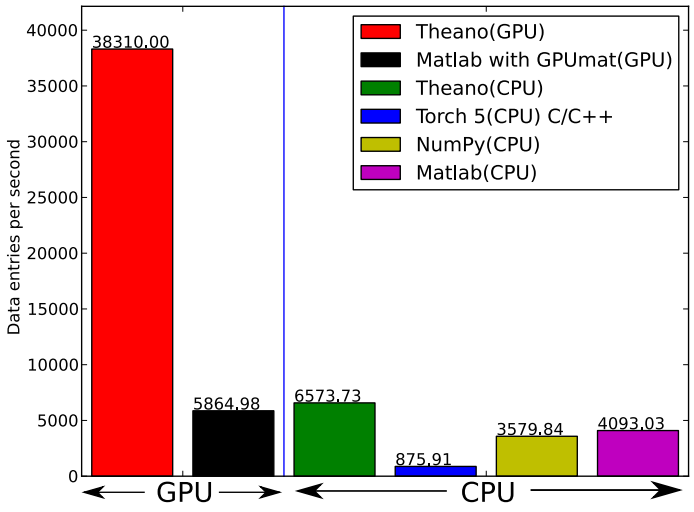

Figure 8.5: Fitting a Multi-Layer Perceptron to simulated data with various implementations of stochastic gradient descent. These models have 784 inputs, 500 hidden units, a 10-way classification, and are trained 60 examples at a time.

Figure 8.5 compares the number of examples processed per second across different implementations. We compared Theano (revision #ec057beb6c) against NumPy 1.4.1, MATLAB 7.9.0.529, and Torch 5 (a machine learning library written

in C/C++) [10] on the CPU and GPUMat 0.25 [11] for MATLAB on the GPU.

When running on the CPU, Theano is 1.8x faster than NumPy, 1.6x faster than MATLAB, and 7.5x faster than Torch 5. Theano's speed increases 5.8x on the GPU from the CPU, a total increase of 11x over NumPy (CPU) and 44x over Torch 5 (CPU). GPUmat brings about a speed increase of only 1.4x when switching to the GPU for the MATLAB implementation, far less than the 5.8x increase Theano achieves through CUDA specializations.

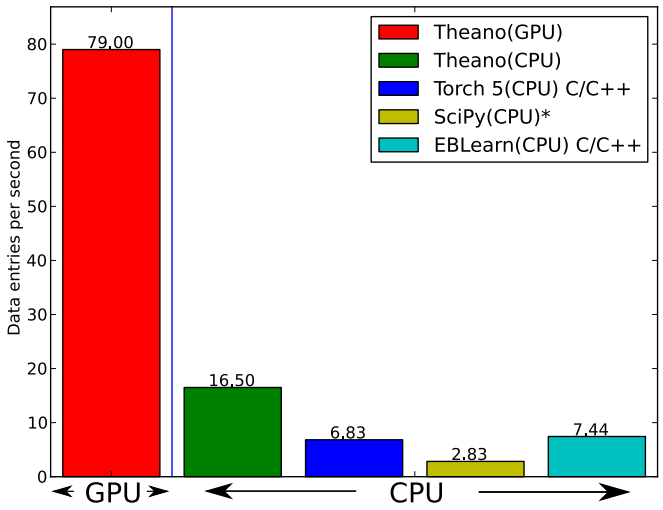

Figure 8.6: Fitting a convolutional network using various software packages. The benchmark stresses convolutions of medium-sized (256 by 256) images with small (7 by 7) filters.

Because of the difficulty in implementing efficient convolutional networks, we only benchmark against known libraries that offer a pre-existing implementation. We compare against EBLearn (Sermanet et al., 2009) and Torch, two libraries written in C++. EBLearn was implemented by members of Yann LeCun's lab at NYU, who have done extensive research in convolutional networks. To put these results into perspective, we implemented approximately half (no gradient calculation) of

10. Torch5 (`http://torch5.sourceforge.net`) was designed and implemented with flexibility in mind, not speed (Ronan Collobert, p.c.).

11. GPUmat: GPU toolbox for MATLAB (`http://gp-you.org`)

the algorithm using SciPy's `signal.convolve2d` function. This benchmark uses convolutions of medium sized images (256×256) with small filters (7×7). Figure 8.6 compares the performance of Theano (both CPU and GPU) with that of competing implementations. On the CPU, Theano is 2.2x faster than EBLearn, its best competitor. This advantage is owed to the fact that Theano compiles more specialized convolution routines. Theano's speed increases 4.9x on the GPU from the CPU, a total of 10.7x over EBLearn (CPU). On the CPU, Theano is 5.8x faster than SciPy even though SciPy is doing only half the computations. This is because SciPy's convolution routine has not been optimized for this application.

We also compared Theano with numexpr and NumPy for evaluating element-wise expressions on the CPU (Figure 8.7). For small amounts of data, the extra function-call overhead of numexpr and Theano makes them slower. For larger amounts of data, and for more complicated expressions, Theano is fastest because it uses an implementation specialized for each expression.

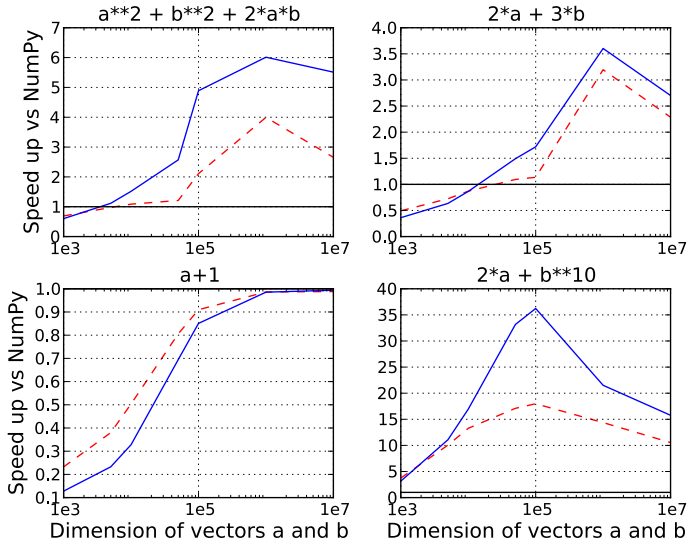

Figure 8.7: CPU Speed comparison between NumPy, numexpr, and Theano for different sizes of input on four element-wise formulae. In each subplot, the solid blue line represents Theano, the dashed red line represent numexpr, and performance is plotted with respect to NumPy.

8.4 What kinds of work does Theano support?

Theano's expression types cover much of the same functionality as NumPy, and include some of what can be found in SciPy. Table 8.I lists some of the most-used expressions in Theano. More extensive reference documentation is available on Theano's website.

Theano's strong suit is its support for strided N-dimensional arrays of integers and floating point values. Signed and unsigned integers of all native bit widths are supported, as are both single-precision and double-precision floats. Single-precision and double-precision complex numbers are also supported, but less so - for example, gradients through several mathematical functions are not implemented. Roughly 90% of expressions for single-precision N-dimensional arrays have GPU implementations. Our goal is to provide GPU implementations for all expressions supported by Theano.

Random numbers are provided in two ways: via NumPy's random module, and via an internal generator from the MRG family (L'Ecuyer et al., 1993). Theano's `RandomStreams` replicates the `numpy.random.RandomState` interface, and acts as a proxy to NumPy's random number generator and the various random distributions that use it. The `MRG_RandomStreams` class implements a different random number generation algorithm (called MRG31k3p) that maps naturally to GPU architectures. It is implemented for both the CPU and GPU so that programs can produce the same results on either architecture without sacrificing speed. The `MRG_RandomStreams` class offers a more limited selection of random number distributions than NumPy though: uniform, normal, and multinomial.

Sparse vectors and matrices are supported via SciPy's `sparse` module. Only compressed-row and compressed-column formats are supported by most expressions. There are expressions for packing and unpacking these sparse types, some operator support (e.g. scaling, negation), matrix transposition, and matrix multiplication with both sparse and dense matrices. Sparse expressions currently have no GPU equivalents.

Table 8.I: Overview of Theano's core functionality. This list is not exhaustive, and is superseded by the online documentation. More details are given in text for items marked with an asterisk. `dimshuffle` is like `numpy.swapaxes`.

Operators	`+`, `-`, `/`, `*`, `**`, `//`, `eq`, `neq`, `<`, `<=`, `>`, `>=`, `&`, `|`, `^`
Allocation	`alloc`, `eye`, `[ones,zeros]_like`, `identity{_like}`
Indexing*	basic slicing (see `set_subtensor` and `inc_subtensor` for slicing lvalues); limited support for advanced indexing
Mathematical Functions	`exp`, `log`, `tan[h]`, `cos[h]`, `sin[h]`, `real`, `imag`, `sqrt`, `floor`, `ceil`, `round`, `abs`
Tensor Operations	`all`, `any`, `mean`, `sum`, `min`, `max`, `var`, `prod`, `argmin`, `argmax`, `reshape`, `flatten`, `dimshuffle`
Conditional	`cond`, `switch`
Looping	`Scan`
Linear Algebra	`dot`, `outer`, `tensordot`, `diag`, `cholesky`, `inv`, `solve`
Calculus*	`grad`
Signal Processing	`conv2d`, `FFT`, `max_pool_2d`
Random	`RandomStreams`, `MRG_RandomStreams`
Printing	`Print`
Sparse	compressed row/col storage, limited operator support, `dot`, `transpose`, conversion to/from dense
Machine Learning	sigmoid, softmax, multi-class hinge loss

There is also support in Theano for arbitrary Python objects. However, there are very few expressions that make use of that support because the compilation pipeline works on the basis of inferring properties of intermediate results. If an intermediate result can be an arbitrary Python object, very little can be inferred. Still, it is occasionally useful to have such objects in Theano graphs.

Theano has been developed to support machine learning research, and that has motivated the inclusion of more specialized expression types such as the logistic sigmoid, the softmax function, and multi-class hinge loss.

8.5 Compilation by `theano.function`

What happens under the hood when creating a function? This section outlines, in broad strokes, the stages of the compilation pipeline. Prior to these stages, the expression graph is copied so that the compilation process does not change anything in the graph built by the user. As illustrated in Figure 8.8, the expression graph is subjected to several transformations: (1) canonicalization, (2) stabilization, (3) specialization, (4) optional GPU transfer, (5) code generation. There is some overlap between these transformations, but at a high level they have different objectives. (The interested reader should note that these transformations correspond roughly, but not exactly to the optimization objects that are implemented in the project source code.)

8.5.1 Canonicalization

The canonicalization transformation puts the user's expression graph into a standard form. For example, duplicate expressions are merged into a single expression. Two expressions are considered duplicates if they carry out the same operation and have the same inputs. Since Theano expressions are purely functional (i.e., cannot have side effects), these expressions must return the same value and thus it is safe to perform the operation once and reuse the result. The symbolic gradient mechanism often introduces redundancy, so this step is quite important.

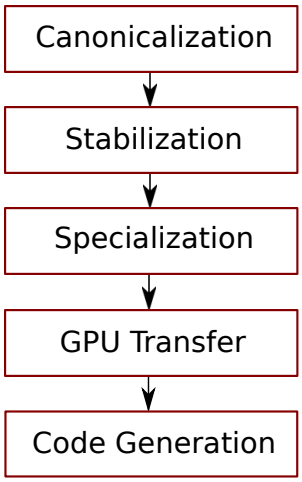

Figure 8.8: The compilation pipeline for functions compiled for GPU. Functions compiled for the CPU omit the GPU transfer step.

For another example, sub-expressions involving only multiplication and division are put into a standard fraction form (e.g. `a / (((a * b) / c) / d) -> (a * c * d) / (a * b) -> (c * d) / (b)`). Some useless calculations are eliminated in this phase, for instance cancelling out uses of the `a` term in the previous example, but also reducing `exp(log(x))` to `x`, and computing outright the values of any expression whose inputs are fully known at compile time. Canonicalization simplifies and optimizes the graph to some extent, but its primary function is to collapse many different expressions into a single normal form so that it is easier to recognize expression patterns in subsequent compilation stages.

8.5.2 Stabilization

The stabilization transformation improves the numerical stability of the computations implied by the expression graph. For instance, consider the function `log(1 + exp(x))`, which tends toward zero as $\lim_{x \to -\infty}$, and `x` as $\lim_{x \to -\infty}$. Due to limitations in the representation of double precision numbers, the computation as written yields infinity for `x > 709`. The stabilization phase replaces patterns like one with an expression that simply returns `x` when `x` is sufficiently large (using doubles, this is accurate beyond the least significant digit). It should be noted that

this phase cannot guarantee the stability of computations. It helps in some cases, but the user is still advised to be wary of numerically problematic computations.

8.5.2.1 Specialization

The specialization transformation replaces expressions with faster ones. Expressions like `pow(x,2)` become `sqr(x)`. Theano also performs more elaborate specializations: for example, expressions involving scalar-multiplied matrix additions and multiplications may become BLAS General matrix multiply (GEMM) nodes and `reshape`, `transpose`, and `subtensor` expressions (which create copies by default) are replaced by constant-time versions that work by aliasing memory. Expressions subgraphs involving element-wise operations are fused together (as in numexpr) in order to avoid the creation and use of unnecessary temporary variables. For instance, if we denote the `a + b` operation on tensors as `map(+, a, b)`, then an expression such as `map(+, map(*, a, b), c)` would become `map(lambda ai,bi,ci: ai*bi+ci, a, b, c)`. If the user desires to use the GPU, expressions with corresponding GPU implementations are substituted in, and transfer expressions are introduced where needed. Specialization also introduces expressions that treat inputs as workspace buffers. Such expressions use less memory and make better use of hierarchical memory, but they must be used with care because they effectively destroy intermediate results. Many expressions (e.g. GEMM and all element-wise ones) have such equivalents. Reusing memory this way allows more computation to take place on GPUs, where memory is at a premium.

8.5.3 Moving Computation to the GPU

Each expression in Theano is associated with an implementation that runs on either the host (a host expression) or a GPU device (a GPU expression). The GPU-transfer transformation replaces host expressions with GPU expressions. The majority of host expression types have GPU equivalents and the proportion is always growing.

The heuristic that guides GPU allocation is simple: if any input or output of an expression resides on the GPU and the expression has a GPU equivalent, then the GPU equivalent is substituted in. Shared variables storing `float32` tensors default to GPU storage, and the expressions derived from them consequently default to using GPU implementations. It is possible to explicitly force any `float32` variable to reside on the GPU, so one can start the chain reaction of optimizations and use the GPU even in graphs with no shared variables. It is possible (though awkward, and discouraged) to specify exactly which computations to perform on the GPU by disabling the default GPU optimizations.

Tensors stored on the GPU use a special internal data type with an interface similar to the `ndarray`. This data type fully supports strided tensors, and arbitrary numbers of dimensions. The support for strides means that several operations such as the transpose and simple slice indexing can be performed in constant time.

8.5.4 Code Generation

The code generation phase of the compilation process produces and loads dynamically-compiled Python modules with specialized implementations for the expressions in the computation graph. Not all expressions have C (technically C++) implementations, but many (roughly 80%) of Theano's expressions generate and compile C or CUDA code during `theano.function`. The majority of expressions that generate C code specialize the code based on the dtype, broadcasting pattern, and number of dimensions of their arguments. A few expressions, such as the small-filter convolution (`conv2d`), further specialize code based on the size the arguments will have.

Why is it so important to specialize C code in this way? Modern x86 architectures are relatively forgiving of code that does not make good use techniques such as loop unrolling and prefetching contiguous blocks of memory, and only the `conv2d` expression goes to any great length to generate many special case implementations for the CPU. By comparison, GPU architectures are much less forgiving of code that is not carefully specialized for the size and physical layout of func-

tion arguments. In response, the code generators for GPU expressions like `GpuSum`, `GpuElementwise`, and `GpuConv2d` generate a wider variety of implementations than their respective host expressions. With the current generation of graphics cards, the difference in speed between a naïve implementation and an optimal implementation of an expression as simple as matrix row summation can be an order of magnitude or more. The fact that Theano's GPU `ndarray`-like type supports strided tensors makes it even more important for the GPU code generators to support a variety of memory layouts. These compile-time specialized CUDA kernels are integral to Theano's GPU performance.

8.6 Limitations and Future Work

While most of the development effort has been directed at making Theano produce fast code, not as much attention has been paid to the optimization of the compilation process itself. At present, the compilation time tends to grow superlinearly with the size of the expression graph. Theano can deal with graphs up to a few thousand nodes, with compilation times typically on the order of seconds. Beyond that, it can be impractically slow, unless some of the more expensive optimizations are disabled, or pieces of the graph are compiled separately.

A Theano function call also requires more overhead (on the order of microseconds) than a native Python function call. For this reason, Theano is suited to applications where functions correspond to expressions that are not too small (see Figure 8.7).

The set of types and operations that Theano provides continues to grow, but it does not cover all the functionality of NumPy and covers only a few features of SciPy. Wrapping functions from these and other libraries is often straightforward, but implementing their gradients or related graph transformations can be more difficult. Theano does not yet have expressions for sparse or dense matrix inversion, nor linear algebra decompositions, although work on these is underway outside of the Theano trunk. Support for complex numbers is also not as widely imple-

mented or as well-tested as for integers and floating point numbers. NumPy arrays with non-numeric dtypes (strings, Unicode, Python objects) are not supported at present.

We expect to improve support for advanced indexing and linear algebra in the coming months. Documentation online describes how to add new operations and new graph transformations. There is currently an experimental GPU version of the scan operation, used for looping, and an experimental lazy-evaluation scheme for branching conditionals.

The library has been tuned for expressions related to machine learning with neural networks, and it is not as well tested outside of this domain. Theano is not a powerful computer algebra system, and it is an important area of future work to improve its ability to recognize numerical instability in complicated element-wise expression graphs.

Debugging Theano functions can require non-standard techniques and Theano specific tools. The reason is two-fold: 1) definition of Theano expressions is separate from their execution, and 2) optimizations can introduce many changes to the computation graph. Theano thus provides separate execution modes for Theano functions, which allows for automated debugging and profiling. Debugging entails automated sanity checks, which ensure that all optimizations and graph transformations are safe (Theano compares the results before and after their application), as well as comparing the outputs of both C and Python implementations.

We plan to extend GPU support to the full range of C data types, but only float32 tensors are supported as of this writing. There is also no support for sparse vectors or matrices on the GPU, although algorithms from the CUSPARSE package should make it easy to add at least basic support for sparse GPU objects.

8.7 Conclusion

Theano is a mathematical expression compiler for Python that translates high level NumPy-like code into machine language for efficient CPU and GPU compu-

tation. Theano achieves good performance by minimizing the use of temporary variables, minimizing pressure on fast memory caches, making full use of `gemm` and `gemv` BLAS subroutines, and generating fast C code that is specialized to sizes and constants in the expression graph. Theano implementations of machine learning algorithms related to neural networks on one core of an E8500 CPU are up to 1.8 times faster than implementations in NumPy, 1.6 times faster than MATLAB, and 7.6 times faster than a related C++ library. Using a Nvidia GeForce GTX285 GPU, Theano is an additional 5.8 times faster. One of Theano's greatest strengths is its ability to generate custom-made CUDA kernels, which can not only significantly outperform CPU implementations but alternative GPU implementations as well.

Acknowledgements

Theano has benefited from the contributions of many members of the machine learning group in the computer science department (Départment d'Informatique et de Recherche Operationelle) at l'Université de Montréal, especially Arnaud Bergeron, Thierry Bertin-Mahieux, Olivier Delalleau, Douglas Eck, Dumitru Erhan, Philippe Hamel, Simon Lemieux, Pierre-Antoine Manzagol, and François Savard. The authors acknowledge the support of the following agencies for research funding and computing support: NSERC, RQCHP, CIFAR, SHARCNET and CLUMEQ.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

This dissertation examined how to incorporate knowledge of the visual system's complex cells into artificial neural networks for image classification. Chapter 3 showed that randomly initialized models based on visual area V1's *complex cells* generalize better from labelled training examples than conventional neural networks whose hidden units are more like V1's *simple cells*. Chapters 4, 5, and 6 developed interpretations for complex-cell-based visual system models as probability distributions and presented tractable algorithms for fitting those distributions to unlabelled data. Taken together, these results show that complex cell models are a better foundation for visual object discrimination than simple cell models, and that unsupervised learning methods can tune them as discriminative features by modelling images.

This research demanded two auxiliary technical innovations: an algorithm for selecting model hyper-parameters, and software that could provide symbolic differentiation and leverage both CPU and GPU architectures. Chapter 7 presented random search as an algorithm for model selection, and argued, based on the idea that hyper-parameter response functions have low effective dimension, that random search is more efficient than grid search. Chapter 8 described Theano, an optimizing compiler written in Python, which provides symbolic differentiation and generates optimized C code for model training and testing routines on the CPU or GPU. Both of these tools have been useful to me, as well as the other members of the learning group at the University of Montreal. Now that they have been published, I hope they will help others outside the lab who are engaged in similar pursuits.

The following sections address open questions for future work to explore.

9.1 Hyper-parameter Optimization

As we explore models with increasing numbers of hyper-parameters, it becomes more important to formalize the practice of hyper-parameter optimization. Random search is a first step, but sequential methods such as those of Srinivasan and Ramakrishnan (2011) and Hutter et al. (2011) offer much greater promise. The key to the success of these sequential methods is the estimation of a *response surface* as results come in, which predicts performance at points that have not yet been tried. In Chapter 7 I used a Gaussian Process to model the response surface, but any other prediction method could be considered instead. One particularity to this prediction problem is that often hyper-parameters are known to be irrelevant, based on the value of other hyper-parameters (e.g. the number of PCA components to use is irrelevant when the preprocessing algorithm is not PCA). Future work will explore how to incorporate this knowledge into prediction algorithms.

9.2 Sum Pooling vs. Max Pooling

Chapter 3 looked at a number of complex-cell-like parameterizations in terms of their ability to support object discrimination by a linear classifier. Two findings from those experiments were that (a) an exponent of 2 on rectified linear responses was better than 1 or 3 (where 3 is more like max-pooling), and (b) for any given number of linear filters, performance was better when they were grouped into pools. Both of these findings are consistent with those of Hyvärinen and Köster (2007), which looked at modelling images using subspace ICA methods (strictly unsupervised learning). The first finding (a) suggests that the max-pooling in convolutional neural networks such as LeCun et al. (1998a) and Riesenhuber and Poggio (1999) might be less effective than sum-of-squares pooling, but additional empirical comparisons on various datasets would be necessary to form a conclusion regarding which is generally better. The second conclusion (b) is potentially inconsistent with the CIFAR-10 results obtained during hyper-parameter optimization of Chapter 5 on the ssRBM. In that search, I tried models of 100, 200, and 300 hidden units

pooling over 1, 2, or 3 linear filters each, and found that 100 hidden units pooling over 3 linear filters performed less well than 300 hidden units pooling over 1 filter each. The evidence for inconsistency is weak because the trials of Chapters 3 and 5 involved different datasets, different models, and different classifiers, but the μ-ssRBM is a good model on which to base an experiment that explores this inconsistency. It might be that pre-training does more good in single-filter hidden units than multi-filter hidden units, or it might be the case that the CIFAR-10 dataset is unlike the datasets used in Chapter 3. Max-pooling can be implemented in ssRBMs by partitioning the binary (spike) hidden variables into groups, in which at most one unit may be on at a time. This contrasts sum-pooling (Chapter 5), which occurs across groups of slab variables. It would be interesting to come back to the question of pool size and the utility of sum-pooling versus max-pooling in μ-ssRBMs, in which the size of sum-pooling groups and max-pooling groups can be manipulated independently.

9.3 Scaling with Nested Models

The convolutional architecture presented in Chapter 6 is one option for scaling ssRBM models up to larger images, but another related architecture that borrows something from SIFT features and dynamic routing (Grimes and Rao, 2005; Olshausen et al., 1993; Tenenbaum and Freeman, 2000) will offer faster learning and a more valuable structured image representation.

Suppose that $E^{(0)}(v,s,h)$ denotes the energy of a μ-ssRBM defined for an image-patch s, a vector of spike variables h, and slab variables s (as described in Chapter 6). Whereas the pooled ssRBM is an energy model of three sets of random variables (v,s,h), the *nested ssRBM* (nssRBM) is a model of five sets of random variables (v,s,h,u,g). The u and g are an additional set of spike and slab variables that will be nested within the hidden variables s and h. Supposing that the nssRBM has R hidden units h, each one ($h_r \in \{0,1\}$) of which multiplies some corresponding

slab vector $s_r \in \mathbb{R}^L$, the energy function of the nssRBM is

$$E^{(1)}(v,s,h,u,g) = \sum_{r=1}^{R} v' T_r s_r h_r + E^{(0)}(s_r,u_r,g_r)h_r. \tag{9.1}$$

Each transformation T_r is a linear one that reduces (in inference from v) the potentially large number of pixels in v down to something resembling an image patch in s_r. When all the T_r serve simply to extract image patches, this is similar (except for the use of h) to the convolutional model presented in Chapter 6. But each T_r need not simply extract pixel patches, it can be parametrized to fetch sub-images at various rotations and scales, as well as non-affine shapes - perhaps even arbitrary linear transforms. My hope is that the various T_r represent the space of features considered by the SIFT algorithm. As in a convolutional architecture, we can reuse the same patch model $E^{(0)}$ to recognize familiar shapes under the variety of transformations. We can even use the same learning principles that train the rest of the model to *learn* the transformation matrices T_r. If the T_r matrices are themselves parametrized by amounts of translation, rotation, and scaling, then those parameters become latent variables of the model that can themselves be sampled or optimized for individual examples for even better likelihood. Except for these latent parameters of each T_r the *nested ssRBM* adds something like depth without compromising the tractability of conditional inference.

Preliminary experiments on this sort of model based on sparse coding, rather than RBM learning, were submitted to NIPS 2010 (Bergstra et al., 2010b). With $E^{(0)}$ in Eqn. 9.1 implemented by a single Gaussian in s_r with mean μ (rather than an ssRBM), we sampled a fixed set of $R = 1000$ transformation matrices T_r to span a range of positions, scales, and orientations, and used the sparse coding algorithm of Olshausen and Field (1996) to learn a *factored sparse coding* dictionary formed by transforming a single *source filter* μ into 1000 random positions, scales, and orientations. A model trained from 5000 greyscale CIFAR-10 images is illustrated in Figure 9.1. Our point in Bergstra et al. (2010b) was that normal (un-factored) sparse coding models cannot learn such a good dictionary from so few images.

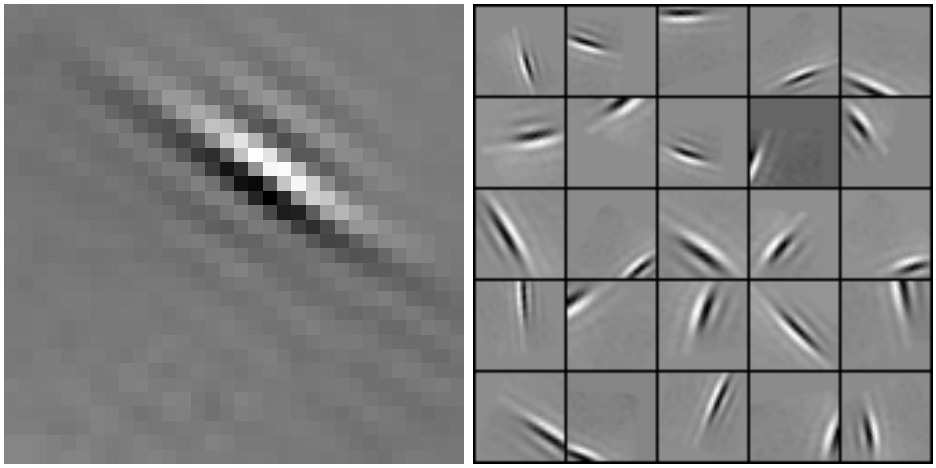

Figure 9.1: The single source filter and a subset of the *factored sparse coding* dictionary of 1000 elements learnt from just 5000 greyscale CIFAR-10 images.

Another result, not discussed in that manuscript, was that a grid (lattice) allocation of transformations in location, scale, and orientation $\boldsymbol{T}_r$ is much less efficient than a randomized allocation. These two results both speak to the tractability of learning in the nested ssRBM model. There is often a desire to learn everything in a model so that it fits the data as well as possible, but there are advantages to restricting $\boldsymbol{T}_r$ to rigid spatial transformations. When $\boldsymbol{T}_r$ is parametrized by amounts of translation, scaling, and rotation, it is easy to visualize the second-layer components of the model. It is more natural to put a prior over rigid transformations, so that the transformations themselves become variables in the model. These coordinates also suggest an intriguing possibility of identifying hidden units not by an index, but instead by the amount of rotation, of translation, and of scaling that they effectively apply to the input. Perhaps this style of encoding can eventually lead to more structured parameterizations of complex shapes (e.g. "there is a circle-type shape a bit to the left of a line-type shape").

9.4 Slow Features in Energy Models

The principle of slowness is appealing, but I feel that the combination of subspace ICA-like algorithm with slow-feature regularization used in Chapter 4 is dif-

ficult to combine with the energy-based approach in the ssRBM. The decorrelation term was necessary in a model with explicit quadratic terms such as that of Rust et al. (2005), because drawing negative samples in such a model is computationally expensive. In contrast, the ssRBM uses implicit quadratic terms and support efficient Gibbs sampling. Thus the decorrelation part of the training algorithm from Chapter 4 is unnecessary when using an ssRBM, but the slowness criterion is still interesting.

Unlike earlier examples of slow feature learning based on regularization, energy-based models (like other probability models) can actually *learn* slowness from data. Any binary latent variable h in an energy model of images can be turned into a *slow feature* by an energy term such as the one in Eqn. 9.2,

$$E^{(\text{slow})}(h^{(1)}, h^{(2)}) = [h^{(1)} h^{(2)}]' S [h^{(1)} h^{(2)}]. \tag{9.2}$$

Slowness requires training on pairs of images rather than individual images. In Eqn. 9.2, $h^{(1)} \in \{0,1\}$ is a binary hidden unit in the first image, $h^{(2)} \in \{0,1\}$ is the same binary hidden unit in the second image, and $S \in \mathbb{R}^{2\times 2}$ encodes the energy contribution of each possible configurations of $h^{(1)}$ and $h^{(2)}$. A negative diagonal and a positive off-diagonal in S encourage unit h to take the same value in both images.

9.5 Complex Cell Models and Depth

The term *deep learning* refers both to the practice of combining unsupervised learning (pretraining) with supervised learning (finetuning), and to the use of that practice to train multilayer models. Deep learning architectures such as those of Hinton et al. (2006); Kavukcuoglu et al. (2010); Lee et al. (2009); Ranzato et al. (2008); Salakhutdinov and Hinton (2009); Vincent et al. (2008) use layers whose units operate in a manner consistent with V1 simple cells. The second layer units in these multilayer models can be seen as pooling together the activations of first layer units, and thus having the capacity to operate like complex cell models. Hubel and

Wiesel (1962) as well as many others since have hypothesized that complex cells do act on input from simple cells. More recently though, the distinction between simple and complex cells has come under dispute (Finn and Ferster, 2007; Kouh and Poggio, 2008; Rust et al., 2005), and by corollary, the idea that simple ones feed into complex ones. Consequently, I think it is both more useful and more accurate to think of deep architectures for vision in terms of the connectivity between layers and the type of each layer, which can vary independently. My work has explored different types of layers, but not different numbers of them. Many approaches in the literature to stacking RBMs and DAEs extend naturally to stacking ssRBMs. The empirical question of how well these approaches work in various applications is an interesting one for future work.

9.6 Revisiting the Feed-forward Model

In Chapter 2 (Section 2.3.6, "Time, Feedback and Learning in the Visual System") The feed-forward rate model hypothesis is a statement about how visual cortex infers the patterns of activity that support effective decision making. It is a statement about what aspects of neural function in the pathway from the retina to the visual cortex and beyond are irrelevant to a mathematical understanding of the perception and judgement of visual input: that the precise timing of spikes is of limited or no importance; that the numerous backprojections are of limited or no importance; that the temporal nature of retinal input is of limited or no importance. Any or all of these simplifying assumptions could be false, which could have various implications for our mathematical and computational understanding of vision.

For the precise timing of spikes to be important, it would have to be the case that neurons do in fact organize their firing patterns among a population in ways that are awkward or impossible to express in rate models (Gray, 1999). The challenge of artificial vision starts with the mystery of how to make sense of vast pixel arrays. If it could be demonstrated that the spike patterns of retinal ganglion cells

were temporally structured as a population, it could provide modellers with insight as to *why* and *how* spike timing is relevant. On the other hand, if spike timing is mainly important for its role in Hebbian learning via spike time dependent plasticity, then spike timing may just be a artifact of the neural implementation of learning algorithms and data structures for representing stimuli that can be implemented perfectly well in rate models. From a modeling perspective, the successes of various rate-based models together with the availability of algorithms for optimization and statistical learning continue to make rate-based models more appealing.

The role of backprojections in inference (short time scale neural activity patterns) is a subject of ongoing debate. On the feed-forward side of the debate we have models such as Ferster and Miller (2000); Fukushima (1980, 2007); LeCun et al. (1998a); Riesenhuber and Poggio (1999) in which feedback is perhaps used for learning, but not for inference at all. On the other extreme are models in which lateral connections and feedback play a dominant role in inference (Hawkins and Blakeslee, 2002; Johson and Olshausen, 2003; Karklin and Lewicki, 2003; Lee and Mumford, 2003; Murray et al., 2002; Olshausen and Field, 1996; Rozell et al., 2008; Salakhutdinov and Hinton, 2009). In these models backprojections are invoked to implement inference in probability models where the the posterior distribution over latent variables is not factorial. There are hybrid models such as predictive sparse decomposition (Ranzato et al., 2008) that present a feed-forward processing path as acting as a fast approximate surrogate for a slower, more correct inference procedure that can be used directly once it has finished, and serves as a training signal for the fast path. There are also hybrid models such as ARTSCAN and ARTSCENE that include several feed-forward components, and a few points of feedback that would use backprojections in a neural implementation (Fazl et al., 2009; Grossberg, 1987; Grossberg and Huang, 2009). These models invoke back projections to implement an attentional mechanism that up-regulates or down-regulates the activities of intermediate variables (neurons). No backprojections are required for inference in the models I've presented in this dissertation, but they would be required in future work that extends these models to larger systems with

more latent variables.

The role of backprojections in learning is also still unclear. The supervised learning algorithms described in this thesis represent one theory for why backprojections exist: namely to transport a gradient signal from later stages in the calculation of a cost function back to the synaptic weights involved with earlier stages of the computation. The biological plausibility of the error backpropagation algorithm has been a subject of debate since its introduction twenty five years ago (Arbib, 2003; Rumelhart et al., 1986a). The unsupervised learning algorithms described in this thesis depend equally on the backward propagation of information, though not the backpropagation of an error gradient specifically. The negative-phase sampling involved in fitting RBM-like models requires computations to proceed both from visible to hidden variables and *vice versa*. There is some support from the reciprocal nature of neural connectivity that cells in cortex could be implementing some sort of blocked Gibbs sampling, but it remains a question for future neurophysiological exploration (Ungerleider and Pessoa, 2008).

I think the main reason that the temporal aspect of vision has been ignored is that computational modelling experiments call for simulations that have, in the past, been too computationally demanding to be practical. There is evidence that we are able to recognize objects in a scene from a still glimpse (Hung et al., 2005), but when it comes to learning, there is evidence that eye movements (if not also a moving environment) are critical to the development of the visual system (Buisseret, 1995; Buisseret et al., 1978). It may be that motion provides the strongest cues to learning structure in the world, and whatever ability we have to recognize objects from glimpses is merely a by-product of our learning of that structure. Work on slow feature learning begins to explore this possibility, but there other possibilities and I look forward to looking at supervised and unsupervised learning problems on video in future work.

If if it important to learn from a temporal signal, what sort of video should we study? If we would like to study the development of the visual system, we should study the input received by the visual system. It's not so hard to image

obtaining such data: one could attach a camera to a newborn's forehead and record its entire lifetime of visual stimulation. With this data, artificial systems could be pitted against the developing animal on identical perception tasks at many stages as the animal matures into an adult. Engineers and scientists are great at making incremental improvements toward a quantitatively defined objective; we should strive to define an objective that is as relevant as possible to the understanding of how the brain learns to see.

Although it is certainly important to apply our modelling effort to the right sort of temporal data, it may also be the case that no video alone is sufficient to train an artificial visual system. Several behavioural and electrophysiological results on cats with and without motor control suggest that even the entire history of visual input to a developing animal is insufficient for the development of a normal visual system and a normal capacity for visually guided behaviour (Hein et al., 1979; Held and Hein, 1963). These studies, suggest that is crucial that the visual system receive proprioceptive feedback from the sensorimotor system regarding what the animal (particularly its eyes) are *doing*, so that the brain can develop an understanding of how its actions affect its perception. Perhaps this propagation of this feedback is part of the reason for so many backprojections in the visual system. To study this sort of development with computer models, we must use real agents in real environments (autonomous robots with cameras), or else artificial agents in computer-rendered worlds. Algorithms for reinforcement learning are well developed, perhaps not to the same level as linear classification algorithms, but I think to a level that we can use reinforcement learning algorithms to test the representations inferred by increasingly effective unsupervised models (Sutton and Barto, 1998). On the neuroscience side, behavioural experiments are much easier to perform than intracranial recordings, which anyway after many decades leave many questions unanswered regarding the non-linear behaviour of individual neurons and the role of plasticity (Olshausen and Field, 2005). Alan Turing described intelligence in terms of behaviour indistinguishable from that of an intelligent agent, and I think we should bear that in mind more literally as we continue to search for

artificial intelligence by machine learning.

BIBLIOGRAPHY

Y. S. Abu-Mostafa. The Vapnik-Chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1(3):312–317, 1989. doi: 10.1162/neco.1989.1.3.312.

E. H. Adelson and J. R. Bergen. Spatiotemporal energy models for the perception of motion. *Journal of the Optical Society of America*, 2(2):284–299, 1985.

E. L. Allwein, R. E. Shapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

F. Alted. Why modern CPUs are starving and what can be done about it. *Computing in Science and Engineering*, 12(2):68–71, 2010.

J.R. Anderson. *The Adaptive Character of Thought*. Lawrence Erlbaum Associates, 1990.

I. A. Antonov and V. M. Saleev. An economic method of computing LP_τ-sequences. *USSR Computational Mathematics and Mathematical Physics*, 19(1):252–256, 1979.

M. A. Arbib. *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2003.

H. B. Barlow. Possible principles underlying the transformation of sensory messages. *Sensory Communication*, pages 217–234, 1961.

M. F. Bear, B. Connors, and M. Paradiso. *Neuroscience: Exploring the Brain*. Lippincott Williams & Wilkins, Hagerstown, MD, 2007. ISBN 0-7817-6003-8.

S. Becker and G. E. Hinton. Learning mixture models of spatial coherence. *Neural Computation*, 5(2):267–277, 1993.

S. Becker, S. Thrun, and K. Obermayer, editors. *Advances in Neural Information Processing Systems 15 (NIPS'02)*, Cambridge, MA, 2003. MIT Press.

S. Behnel, R. W. Bradshaw, and D. S. Seljebotn. Cython tutorial. In G. Varoquaux, S. van der Walt, and J. Millman, editors, *Proceedings of the 8th Python in Science Conference*, pages 4 – 14, Pasadena, CA USA, 2009.

A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129–1159, 1995.

A. J. Bell and T. J. Sejnowski. The 'independent components' of natural scenes are edge filters. *Vision Research*, 37:3327–3338, 1997.

R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, New Jersey, 1961.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. doi: 10.1561/2200000006.

Y. Bengio and X. Glorot. Understanding the difficulty of training deep feedforward neural networks. In Teh and Titterington (2010), pages 249–256.

Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.

Y. Bengio, O. Delalleau, and N. Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS'05)*, pages 107–114, Cambridge, MA, 2006. MIT Press.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In Schölkopf et al. (2007), pages 153–160.

Y. Bengio, D. Schuurmans, C. Williams, J Lafferty, and A. Culotta, editors. *Advances in Neural Information Processing Systems 22 (NIPS'09)*, 2009. MIT Press.

A. C. Berg and J. Malik. Geometric blur and template matching. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'01)*, 2001.

J. Bergstra and Y. Bengio. Slow, decorrelated features for pretraining complex cell-like networks. In Bengio et al. (2009), pages 99–107.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 2011. Submitted (11-077).

J. Bergstra, G. Desjardins, P. Lamblin, and Y. Bengio. Quadratic polynomials learn better image features. Technical Report 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, April 2009.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010a. Oral.

J. Bergstra, A. Courville, and Y. Bengio. On the statistical inefficiency of sparse coding. Submitted to Lafferty et al. (2010)., 2010b.

J. Bergstra, M. Mandel, and D. Eck. Scalable genre and tag prediction with spectral covariance. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 507–512, August 2010c.

J. Bergstra, Y. Bengio, and J. Louradour. Suitability of V1 energy models for object classification. *Neural Computation*, 23(3):774–790, March 2011a.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, and Y. Bengio. Deep learning in Python with Theano. Journal of Machine Learning Research, 2011b. In preparation.

P. Berkes and L. Wiskott. Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6):579–602, 2005.

P. Berkes, R. E. Turner, and M. Sahani. A structured model of video reproduces primary visual cortical organizations. *PLoS Computational Biology*, 5(9):e1000495, 2009a. doi: 10.1371/journal.pcbi.1000495.

P. Berkes, B. L. White, and J. Fiser. No evidence for active sparsification in the visual cortex. In Bengio et al. (2009), pages 108–116.

C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, London, UK, 1995.

G. G. Blasdel and G. Salama. Voltage sensitive dyes reveal a modular organization in monkey striate cortex. *Nature*, 321:579–585, 1986.

A. B. Bonds. Role of inhibition in the specification of orientation selectivity of cells in the cat striate cortex. *Vision Neuroscience*, 2:41–55, 1989.

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 89–96. ACM, 2007.

A. Bordes, L. Bottou, and P. Gallinari. SGD-QN: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009. See also Bordes et al. (2010).

A. Bordes, L. Bottou, P. Gallinari, J. Chang, and S. A. Smith. Erratum: SGDQN is less careful than expected. *Journal of Machine Learning Research*, 11: 2229âĹŠ2240, Aug 2010.

A. Bosch, A. Zisserman, and Munoz X. Representing shape with a spatial pyramid kernel. In *International Conference on Image and Video Retrieval (CIVR)*, 2007.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

P. Bratley, B. L. Fox, and H. Niederreiter. Implementation and tests of low-discrepancy sequences. *Transactions on Modeling and Computer Simulation, (TOMACS)*, 2(3):195–213, 1992.

C. E. Bredfeldt and D. L. Ringach. Dynamics of spatial frequency tuning in macaque V1. *Journal of Neuroscience*, 22(5):1976–84, March 2002.

M. Brown and D. G. Lowe. Invariant features from interest point groups. In *British Machine Vision Conference (BMVC)*, pages 656–665, Cardiff, Wales, September 2002.

P. Buisseret. Influence of extraocular muscle proprioception on vision. *Physiology Review*, 75:323–338, 1995.

P. Buisseret, E. Gary-Bobo, and M. Imbert. Ocular motility and recovery of orientation properties of visual cortical neurons in dark-reared kittens. *Nature*, 272: 816–817, 1978.

C. Cadieu. *Probabilistic Models of Phase Variables for Visual Representation and Neural Dynamics*. PhD thesis, University of California, Berkeley, 2009.

C. Cadieu and B. A. Olshausen. Learning transformational invariants from natural movies. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pages 209–216. MIT Press, 2009.

R. E. Caflisch, W. Morokoff, and A. Owen. Valuation of mortgage backed securities using brownian bridges to reduce effective dimension, 1997.

M. Carandini. What simple and complex cells compute. *Journal of Physiology*, 577:463–466, 2006.

M. Carandini and D. J. Heeger. Summation and division by neurons in primate visual cortex. *Science*, 264(5163):1333–1336, May 1994.

M. A. Carreira-Perpiñan and G. E. Hinton. On contrastive divergence learning. In R. G. Cowell and Z. Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

J. R. Cavanaugh, W. Bair, and J. A. Movshon. Nature and interaction of signals from the receptive field center and surround in macaque V1 neurons. *Journal of Neurophysiology*, 88:2530–2546, 2002.

C. Chang and C. Lin. *LIBSVM: A Library for Support Vector Machines*, 2001.

H. Chen and A. F. Murray. A continuous restricted Boltzmann machine with an implementable training algorithm. *IEE Proceedings of Vision, Image and Signal Processing*, 150(3):153–158, 2003. doi: 10.1049/ip-vis:20030362.

Y. Chen, S. Anand, S. Martinez-Conde, S. L. Macknik, Y. Bereshpolova, H. A. Swadlow, and J. Alonso. The linearity and selectivity of neuronal responses in awake visual cortex. *Journal of Vision*, 9(9):1–17, 2009.

A. Coates, H. Lee, and A. Ng. An analysis of single-layer networks in unsupervised feature learning. NIPS Deep Learning and Unsupervised Feature Learning Workshop, 2010.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

C. E. Connor, S. L. Brincat, and A. Pasupathy. Transformation of shape information in the ventral pathway. *Current Opinions in Neurobiology*, 17:140–147, 2007.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

A. Courville, J. Bergstra, and Y. Bengio. The spike and slab restricted Boltzmann machine. NIPS Deep Learning and Unsupervised Feature Learning Workshop, 2010.

A. Courville, J. Bergstra, and Y. Bengio. The spike and slab restricted Boltzmann machine. In *Proc. of The Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, 2011a. Accepted.

A. Courville, J. Bergstra, and Y. Bengio. Unsupervised models of images by spike and slab RBMs. International Conference on Machine Learning (ICML), 2011b. Submitted.

D. D. Cox and J. J. DiCarlo. Does learned shape selectivity in inferior temporal cortex automatically generalize across retinal position? *Neuroscience*, 28(40): 10045–10055, 2008.

D. D. Cox, P. Meier, N. Oertelt, and J. J. DiCarlo. "Breaking" position invariant object recognition. *Nature Neuroscience*, 8:1145–1147, 2005.

D. D. Cox, N. Pinto, D. Doukan, B. Cordas, and J. J. DiCarlo. A high-throughput screening approach to discovering good forms of visual representation. Abstract and poster, COSYNE, 2008.

C. A. Curcio and K. A. Allen. Topography of ganglion cells in human retina. *Journal of Computational Neurology*, 300(1), October 1990.

I. Czogiel, K. Luebke, and C. Weihs. Response surface methodology for optimizing hyper parameters. Technical report, Universität Dortmund Fachbereich Statistik, September 2005.

G. E. Dahl, M. Ranzato, A. Mohamed, , and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In J. Lafferty, C. Williams,

J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 469–477, 2010.

P. Dayan and L. F. Abbott. *Theoretical Neuroscience*. The MIT Press, 2001.

D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002. doi: 10.1023/A:1012454411458.

E. Doi and M. S. Lewicki. A theory of retinal population coding. In *Advances in Neural Information Processing Systems 19*, Cambridge, 2007. MIT Press.

J. Dowling. Retina. *Scholarpedia*, 2(12):3487, 2007. doi: 10.4249/scholarpedia.3487.

S. S. Drew and T. Homem de Mello. Quasi-monte carlo strategies for stochastic optimization. In *Proc. of the 38th Conference on Winter Simulation*, pages 774 – 782, 2006.

S. Edelman, N. Intrator, and T. Poggio. Complex cells and object recognition. Unpublished, 1997.

D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, pages 153–160, Clearwater, FL, 2009.

D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

M. Galassi et al. *GNU Scientific Library Reference Manual*, 3rd edition, 2009.

B. F. Farley, H. Yu, D. Z. Jin, and M. Sur. Alteration of visual input results in a coordinated reorganization of multiple visual cortex maps. *Journal of Neuroscience*, 27:10299–10310, 2007.

A. Fazl, S. Grossberg, and E. Mingolla. View-invariant object category learning, recognition, and search: How spatial and object attention are coordinated using surface-based attentional shrouds. *Cognitive Psychology*, 58:1–48, 2009.

D. Ferster and K. D. Miller. Neural mechanisms of orientation selectivity in the visual cortex. *Annual Reviews in Neuroscience*, 23:441–471, 2000.

D. J. Field. What is the goal of sensory coding? *Neural Computation*, 6:559–601, 1994.

I. M. Finn and D. Ferster. Computational diversity in complex cells of cat primary visual cortex. *Journal of Neuroscience*, 27(36):9638–9648, September 2007.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(111), 1936.

V. H. Franz, K. R. Gegenfurtner, H. H. Bülthoff, and M. Fahle. Grasping visual illusions: No evidence for a dissociation between perception and action. *Psychological Science*, 11(1):20–25, January 2000. doi: 10.1111/1467-9280.00209.

M. Franzius, H. Sprekeler, and L. Wiskott. Slowness and sparseness lead to place, head-direction, and spatial-view cells. *PLoS Computational Biology*, 3(8):e166, 2007.

Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical Report UCSC-CRL-94-25, University of California, Santa Cruz, 1994.

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

K. Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

J. L. Gallant, C. E. Connor, S. Rakshit, J. W. Lewis, and D. C. Van Essen. Neural responses to polar, hyperbolic, and Cartesian gratings in area V4 of the macaque monkey. *Journal of Neurophysiology*, 76:2718–2739, 1996.

M. A. Goodale and A. D. Milner. Separate visual pathways for perception and action. *Trends in Neuroscience*, 15(1):20–25, January 1992. doi: doi:10.1016/0166-2236(92)90344-8.

C. M. Gray. The temporal correlation hypothesis of visual feature integration: Still alive and well. *Neuron*, 24:31–47, 1999.

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 399–406. ACM, 2010.

D. B. Grimes and R. P.N. Rao. Bilinear sparse coding for invariant vision. *Neural Computation*, 17(1):47 – 73, 2005.

S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63, 1987.

S. Grossberg and T.-R. Huang. ARTSCENE: A neural system for natural scene classification. *Vision*, 9(4):1–19, 2009. doi: 10.1167/9.4.6.

R. Grosse, R. Raina, H. Kwong, and A. Y. Ng. Shift-invariant sparse coding for audio classification. In *Proceedings of the 23th Conference in Uncertainty in Artificial Intelligence (UAI'07)*, 2007.

J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

E. J. Hartman, J. D. Keeler, and J. M. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2(2): 210–215, 1990.

M. Haüsser and B. Mel. Dendrites: Bug or feature? *Current Opinion in Neurobiology*, 13:372–383, 2003.

J. Hawkins and S. Blakeslee. *On Intelligence*. Times Books, New York, 2002.

D. J. Heeger. Normalization of cell responses in cat striate cortex. *Visual Neuroscience*, 9(2):181–198, 1992.

J. Hegdé and D. C. Van Essen. Selectivity for complex shapes in primate visual area V2. *Journal of Neuroscience*, 20:1–6, 2000.

A. Hein, F. Vital-Durand, W. Salinger, and R. Diamond. Eye movements initiate visual-motor development in the cat. *Science*, 204:1321–1322, 1979.

R. Held and A. Hein. Movement-produced stimulation in the development of visually guided behavior. *Comparative and Physiological Psychology*, 56(5):872–876, 1963.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

G. E. Hinton. To recognize shapes, first learn to generate images. In P. Cisek, T. Drew, and J. Kalaska, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*. Elsevier, 2007.

G. E. Hinton. A practical guide to training restricted Boltzmann machines. Technical Report 2010-003, University of Toronto, 2010. version 1.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

K. Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: 10.1016/0893-6080(89)90020-8.

J. C. Horton. Ocular integration in human visual cortex. *Canadian. Journal of Ophthalmology*, 41:584–593, 2006.

P. O. Hoyer and A. Hyvärinen. Independent component analysis applied to feature extraction from colour and stereo images. *Network: Computation in Neural Systems*, 11(3):191–210, 2000.

D. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–43, 1968.

D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160(1):106–154, 1962.

C. Hung, G. Kreiman, T. Poggio, and J. J. DiCarlo. Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310:863–866, 2005.

J. Hurri and A. Hyvärinen. Temporal coherence, natural image sequences, and the visual cortex. In Becker et al. (2003), pages 141–148.

F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, 2009.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION-5*, 2011. Extended version as UBC Tech report TR-2010-10.

A. Hyvärinen. Statistical models of natural images and cortical visual representation. *Topics in Cognitive Science*, 2:251–264, 2009.

A. Hyvärinen and P. O. Hoyer. A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41(18):2413–2423, 2001.

A. Hyvärinen and U. Köster. Complex cell pooling and the statistics of natural images. *Network: Computation in Neural Systems*, 18:81–100, 2007.

A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4–5):411–430, 2000.

A. Hyvärinen, P. O. Hoyer, and M. Inki. Topographic independent component analysis. *Neural Computation*, 13(7):1527–1558, 2001.

J. S. Johson and B. A. Olshausen. Timecourse of neural signatures of object recognition. *Vision*, 3:499–512, 2003.

H. E. Jones, W. Wang, and A. M. Sillito. Spatial organization and magnitude of orientation contrast interactions in primate V1. *Journal of Neurophysiology*, 88: 2796–2808, 2002.

Y. Karklin and M. S. Lewicki. Learning higher-order structures in natural images. *Network: Computation in Neural Systems*, 14:483–499, 2003.

Y. Karklin and M. S. Lewicki. Emergence of complex cell properties by learning to generalize in natural scenes. *Nature*, 457:83–86, November 2008. doi: 10.1038/nature07481.

S. Kastner and L. G. Ungerleider. Mechanisms of visual attention in the human cortex. *Annual Reviews in Neuroscience*, 23:315–341, 2000.

K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun. Learning invariant features through topographic filter maps. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'09)*. IEEE Press, 2009.

K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierarchies for visual recognition. In *NIPS*, 2010.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

E. Kobatake and K. Tanaka. Neuronal selectivities to complex object features in the ventral visual pathway of the macaque cerebral cortex. *Journal of Neurophysiology*, 71:856–867, 1994.

T. Kohonen. Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics*, 75(4):281–291, 1996. doi: 10.1007/s004220050295.

D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors. *Advances in Neural Information Processing Systems 21 (NIPS'08)*, 2009. MIT Press.

K. P. Körding, C. Kayser, W. Einhäuser, and P. König. How are complex cell properties adapted to the statistics of natural stimuli? *Journal of Neurophysiology*, 91:206–212, 2004.

M. M. Kouh and T. T. Poggio. A canonical neural circuit for cortical nonlinear operations. *Neural Computation*, 20(6):1427–51, 2008.

G. Kreiman, C. P. Hung, A. Kraskov, R. Q. Quiroga, T. Poggio, and J. J. DiCarlo. Object selectivity of local field potentials and spikes in the macaque inferior temporal cortex. *Neuron*, 49:433–445, 2006. doi: 10.1016/j.neuron.2005.12.019.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

A. Krizhevsky. Convolutional deep belief networks on CIFAR-10. Technical report, University of Toronto, 2010.

J. Lafferty, C. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors. *Advances in Neural Information Processing Systems 23*, 2010. MIT Press.

H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 536–543. ACM, 2008.

H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In Lafferty et al. (2010), pages 1243–1251.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 473–480. ACM, 2007.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'06)*, 2006.

N. Le Roux, P. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 849–856. MIT Press, Cambridge, MA, 2008.

Y. LeCun. Une procedure d'apprentissage pour reseau a seuil assymetrique. In *Proc. of Cognitiva 85: A la Frontiere de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris, 1985.

Y. LeCun. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998-.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998a.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and K. Muller, editors, *Neural Networks: Tricks of the Trade*. Springer, 1998b.

Y. LeCun, F. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the Computer Vision*

and Pattern Recognition Conference (CVPR'04), volume 2, pages 97–104, Los Alamitos, CA, USA, 2004. IEEE Computer Society. doi: 10.1109/CVPR.2004.144.

P. L'Ecuyer, F. Blouin, and R. Couture. A search for good multiple recursive generators. *ACM Transactions on Modeling and Computer Simulation*, 3:87–98, 1993.

H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 801–808. MIT Press, 2008a.

H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area V2. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 873–880, Cambridge, MA, 2008b. MIT Press.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, 2009.

T. S. Lee and D. Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, 20(7):1434–1448, 2003.

G. Leuba and R. Kraftsik. Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age. *Anatomy and Embryology*, 190(4):351–366, 1994. doi: 10.1007/BF00187293.

N. Li and J. J. DiCarlo. Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science*, 321(5895):1502–1507, 2008.

G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston,

editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision 2 (ICCV)*, pages 1150–1157, 1999. doi: 10.1109/ICCV.1999.790410.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

S. Lyu and E. P. Simoncelli. Statistically and perceptually motivated nonlinear image representation. In B. E. Rogowitz, T. N. Pappas, and S. J. Daly, editors, *Proc. SPIE, Conf. on Human Vision and Electronic Imaging XII*, volume 6492, pages 67–91. Society of Photo-Optical Instrumentation, 2007. doi: 10.1117/12.729071.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:10–60, 2010.

J. Martens. Deep learning via hessian-free optimization. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 735–742. ACM, 2010.

J. H. R. Maunsell and S. Treue. Feature-based attention in visual cortex. *Trends in Neuroscience*, 29:317–322, 2006.

C. McCollough. Adaptation of edge-detectors in the human visual system. *Science*, 149:1115–1116, 1965.

M. L. Minksy and S. A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

T. J. Mitchell and J. J. Beauchamp. Bayesian variable selection in linear regression. *J. Amer. Statistical Assoc.*, 83(404):1023–1032, 1988.

M. Mobahi, R. Collobert, and J. Weston. Deep learning from temporal coherence in video. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, 2009.

J. A. Movshon, I. D. Thompson, and D. J. Tolhurst. Receptive field organization of complex cells in the cat's striate cortex. *Journal of Physiology*, 283:79–99, 1978.

S. O. Murray, D. Kersten, B. A. Olshausen, P. Schrater, and D. L. Woods. Shape perception reduces activity in human primary visual cortex. *Proceedings of the National Academy of Science*, 99(23):15164–15169, 2002.

J. Mutch and D. G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision (IJCV)*, 80:45–57, 2008.

V. Nair and G. E. Hinton. Implicit mixtures of restricted Boltzmann machines. In Koller et al. (2009), pages 1145–1152.

V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, Haifa, Israel, June 2010. Omnipress.

K. I. Naka and W. A. Rushton. S-potentials from luminosity units in the retina of fish (cyprinidae). *The Journal of physiology*, 185(3):587–99, 1966.

A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. *Applied Optimization*, 86:523–544, 2003.

R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Dept. of Computer Science, University of Toronto, 1994.

R. M. Neal. Assessing relevance determination methods using DELVE. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 97–129. Springer-Verlag, 1998.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

D. Q. Nykamp and D. L. Ringach. Full identification of a linear-nonlinear system via cross-correlation analysis. *Journal of Vision*, 2:1–11, 2002.

T. E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9:10, 2007.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–691, 1996.

B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research*, 37:3311–3325, December 1997.

B. A. Olshausen and D. J. Field. How close are we to understanding V1? *Neural Computation*, 17:1665–1699, 2005.

B. A. Olshausen, C. H. Anderson, and D. C. Van Essen. A neurobiological model of visual attention and invariant pattern recognition based on dynamical routing of information. *Neuroscience*, 13(11):4700–4719, November 1993.

A. Pasupathy and C. E. Connor. Shape representation in area V4: Position-specific tuning for boundary conformation. *Journal of Neurophysiology*, 86:2505–2519, 2001.

B. Pesaran, M. J. Nelson, and R. A. Andersen. Dorsal premotor neurons encode the relative position of the hand, eye, and goal during reach planning. *Neuron*, 51(1):125–34, 2006.

A. Peters, B. R. Payne, and J. Budd. A numerical analysis of the geniculocortical input to striate cortex in the monkey. *Cerebral Cortex*, 4:215–229, 1994.

N. Pinto, D. D. Cox, and J. J. DiCarlo. Why is real-world visual object recognition hard? *PLoS Computational Biology*, 4(1), 2008. doi: doi:10.1371/journal.pcbi.0040027.

N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Comput Biol*, 5(11):e1000579, 11 2009.

K. Popper. *Logik der Forschung*. Julius Springer Verlag, Vienna, 1935.

M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis*, pages 51–67, 1994.

R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 759–766. Omnipress, 2007.

M. Ranzato and G. E. Hinton. Modeling pixel means and covariance using factorized third-order Boltzmann machines. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'10)*. IEEE Press, 2010.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In Schölkopf et al. (2007), pages 1137–1144.

M. Ranzato, Y. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1185–1192. MIT Press, 2008.

M. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-way restricted Boltzmann machines for modeling natural images. In Teh and Titterington (2010).

M. Ranzato, V. Mnih, and G. Hinton. Generating more realistic images using gated MRF's. In Lafferty et al. (2010), pages 2002–2010.

R. P. N. Rao and D. H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1), January 1999.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

P. A. Rhodes. Recoding patterns of sensory input: Higher-order features and the function of nonlinear dendritic trees. *Neural Computation*, 20(8):2000–2036, 2008.

M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

F. Rosenblatt. *Principles of Neurodynamics*. Spartam Books, 1962.

C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 20:2526–2563, 2008.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986a.

D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, 1986b.

N. Rust, O. Schwartz, J. A. Movshon, and E. Simoncelli. Spatiotemporal elements of macaque V1 receptive fields. *Neuron*, 46(6):945–956, 2005.

R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, volume 5, pages 448–455, Clearwater, FL, April 2009.

R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In Z. Ghahramani, editor, *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pages 791–798, New York, NY, 2007. ACM.

T. Sato, T. Kawamura, and E. Iwai. Responsiveness of inferotemporal single units to visual pattern stimuli in monkeys performing discrimination. *Experimental Brain Research*, 38:313–319, 1980.

B. Schölkopf, J. Platt, and T. Hoffman, editors. *Advances in Neural Information Processing Systems 19 (NIPS'06)*, 2007. MIT Press.

N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

N. N. Schraudolph and T. Graepel. Combining conjugate direction methods with stochastic approximation of gradients. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, pages 7–13. Society for Artificial Intelligence and Statistics, 2003.

N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-newton method for online convex optimization. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, pages 436–443. Journal of Machine Learning Research, 2007.

P. Sermanet, K. Kavukcuoglu, and Y. LeCun. EBLearn: Open-source energy-based learning in C++. In *Proc. International Conference on Tools with Artificial Intelligence (ICTAI'09)*. IEEE, 2009.

T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio. A quantitative theory of immediate visual recognition. *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, 165:33–56, 2007a.

T. Serre, A. Oliva, and T. Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences, USA*, 104(15): 6424–6429, 2007b. doi: 10.1073/pnas.0700622104.

L. Shams and C. von der Malsburg. The role of complex cells in object recognition. *Vision Research*, 42(22):2547–2554, 2002.

E. P. Simoncelli and B. A. Olshausen. Natural image statistics and neural representations. *Annual Reviews in Neuroscience*, 24:1193–1216, 2001.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA, 1986.

H. Sprekeler, C. Michaelis, and L. Wiskott. Slowness: An objective for spike-timing-dependent plasticity? *PLoS Computational Biology*, 3(6):e112, June 2007. doi: doi:10.1371/journal.pcbi.0030112.

A. Srinivasan and G. Ramakrishnan. Parameter screening and optimisation for ILP using designed experiments. *Journal of Machine Learning Research*, 12:627–662, February 2011.

I. H. Stevenson, B. Cronin, M. Sur, and K. P. Körding. Sensory adaptation and short term plasticity as Bayesian correction for a changing brain. *PLoS ONE*, 5 (8):e12436, 2010. doi: 10.1371/journal.pone.0012436.

I. Sutskever, G. E. Hinton, and G. Taylor. The recurrent temporal restricted Boltzmann machine. In Koller et al. (2009), pages 1601–1608.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

N. V. Swindale. Visual map. *Scholarpedia*, 3(6):4607, 2008. doi: 10.4249/scholarpedia.4607.

G. Taylor and G. E. Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In L. Bottou and M. Littman, editors, *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, pages 1025–1032, Montreal, June 2009. Omnipress.

Y. W. Teh and M. Titterington, editors. *Proc. of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, 2010.

J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1064–1071. ACM, 2008.

A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

J. Turian, J. Bergstra, and Y. Bengio. Quadratic features and deep architectures for chunking. In *North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL HLT)*, pages 245–248, Boulder, Colorado, June 2009. Association for Computational Linguistics.

A. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–60, 1950.

C. W. Ueberhuber. Minimization methods. In *Numerical Computation (Volume 2)*, pages 325–335. Springer, 1997.

Ungerleider and Mishkin. Two cortical visual systems. In D. J. Ingle, M. A. Goodale, and R. J. W. Mansfield, editors, *Analysis of Visual Behavior*. MIT Press, 1982.

L. G. Ungerleider and L. Pessoa. What and where pathways. *Scholarpedia*, 3(11): 5342, 2008.

V. N. Vapnik. *Estimation of Dependences based on Empirical Data: Addendum 1.* Springer Verlag, 1982.

V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1989.

V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

V. N. Vapnik and A. Chervonenkis. On the uniform convergence of the relative frequencies of events to their probablities. *Theory Prob. Appl.*, 16:264–280, 1971.

V. N. Vapnik, E. Levin, and Y. LeCun. Measuring the VC-dimension of a learning machine. *Neural Computation*, 6(5):851 – 876, 1994. doi: 10.1162/neco.1994.6.5.851.

P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Machine Learning Research*, 11:3371–3408, 2010.

L. Von Melchner, S. L. Pallas, and M. Sur. Visual behavior mediated by retinal projections directed to the auditory pathway. *Nature*, 404:871–876, 2000.

M. J. Wainwright, O. Schwartz, and E. P. Simoncelli. Natural image statistics and divisive normalization: Modeling nonlinearity and adaptation in cortical neurons. In *Probabilistic Models of the Brain: Perception and Neural Function*, pages 203–222. MIT Press, 2002.

T. Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published, second edition, 2009. Online available at http://www.it-weise.de/.

M. Welling, G. E. Hinton, and S. Osindero. Learning sparse topographic representations with products of Student-t distributions. In Becker et al. (2003), pages 1359–1366.

M. Welling, M. Rozen-Zhi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pages 1481–1488, Cambridge, MA, 2005. MIT Press.

J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1168–1175, Helsinki, Finland, 2008. ACM. doi: 10.1145/1390156.1390303.

C. K. I. Williams. Continuous-valued Boltzmann machines. Unpublished Manuscript, March 1993.

C. K. I. Williams. On a connection between kernel PCA and metric multidimensional scaling. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (NIPS'00)*, pages 675–681. MIT Press, 2001.

H. R. Wilson and J. D. Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12(1):1–24, Jan 1972.

L. Wiskott and T. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

R. Wyss, P. König, and P. Verschure. A model of the ventral visual system based on temporal stability and local memory. *PLoS Biology*, 4(120), 2006.

L. Younes. On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates. In *Stochastics and Stochastics Models*, pages 177–228, 1998.

R. S. Zemel, C. K. I. Williams, and M. Mozer. Directional-unit Boltzmann machines. In *Advances in Neural Information Processing Systems 5 (NIPS'92)*, pages 172–179, San Francisco, CA, 1993. Morgan Kaufmann Publishers Inc.

C. Zetzsche, G. Krieger, and B. Wegmann. The atoms of vision: Cartesian or polar? *Journal of the Optical Society of America A*, 16(7):1554–1565, 1999. doi: 10.1364/JOSAA.16.001554.